ML Tutorial

Introduction

In this report, we analyze a dataset related to laptop prices using various machine learning algorithms. The dataset includes attributes such as brand, processor type, RAM size, storage capacity, GPU, screen size, resolution, battery life, weight, operating system, and price. The objective of this assignment is to apply both supervised and unsupervised learning and Ensemble Learning techniques to gain insights and predict outcomes based on these features.

To conduct this analysis, we will be implementing machine learning models using Weka, Python, and R programming. The tutorial focuses on eight different algorithms, categorized under supervised and unsupervised learning and Ensemble Learning .

Supervised Learning Algorithms:

1. **Decision Tree**: A hierarchical model that splits the dataset based on feature values to make predictions.
2. **Naive Bayes**: A probabilistic classifier based on Bayes' theorem, assuming independence between features.
3. **K-Nearest Neighbors (KNN)**: A distance-based classifier that assigns a class label based on the majority vote of its neighbors.
4. **Support Vector Machine (SVM):** A powerful algorithm that finds an optimal hyperplane to separate different classes.

Unsupervised Learning Algorithm:

5. **K-Means:** A clustering algorithm that groups data points into clusters based on similarity.

Ensemble Learning Algorithms:

6. **Random Forest**: An ensemble learning method that builds multiple decision trees and combines their predictions.
7. **AdaBoost**: A boosting algorithm that combines weak learners to improve classification performance.
8. **Stacking**: A meta-learning technique that combines multiple models to improve predictive accuracy.

By implementing these algorithms in different platforms, we aim to compare their performance, understand their strengths and weaknesses, and identify the best model for predicting laptop prices. The results will be evaluated based on various performance metrics such as accuracy, precision, recall, and F1-score for classification tasks.

This study will help in understanding the effectiveness of different machine learning approaches in analyzing real-world datasets and provide valuable insights into the factors influencing laptop prices.

DATASET :

	Brand	Processor	RAM (GB)	Storage	GPU	Screen Size (inch)	Resolution	Battery Life (hours)	Weight (kg)	Operating System	Price ($)
0	Apple	AMD Ryzen 3	64	512GB SSD	Nvidia GTX 1650	17.3	2560x1440	8.9	1.42	FreeDOS	3997.07
1	Razer	AMD Ryzen 7	4	1TB SSD	Nvidia RTX 3080	14.0	1366x768	9.4	2.57	Linux	1355.78
2	Asus	Intel i5	32	2TB SSD	Nvidia RTX 3060	13.3	3840x2160	8.5	1.74	FreeDOS	2673.07
3	Lenovo	Intel i5	4	256GB SSD	Nvidia RTX 3080	13.3	1366x768	10.5	3.10	Windows	751.17
4	Razer	Intel i3	4	256GB SSD	AMD Radeon RX 6600	16.0	3840x2160	5.7	3.38	Linux	2059.83

Q1) Decision Tree

R Programming

Code :

```
# Load required libraries
library(rpart)
library(rpart.plot)
library(caret)
# Load dataset
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"  # Updated file path
data <- read.csv(file_path)
# View dataset structure
str(data)
print(colnames(data))  # Check column names to avoid errors
# Handle column name issues
colnames(data) <- gsub("[^[:alnum:]_]", "_", colnames(data))  # Fix special characters in column names
# Convert categorical columns to factors
data$Brand <- as.factor(data$Brand)
data$Processor <- as.factor(data$Processor)
data$GPU <- as.factor(data$GPU)
data$Operating_System <- as.factor(data$Operating_System)
# Convert Storage to numeric
data$Storage <- as.numeric(gsub("[^0-9]", "", data$Storage))
# Convert Resolution to total pixel count
resolution_split <- strsplit(as.character(data$Resolution), "x")
```

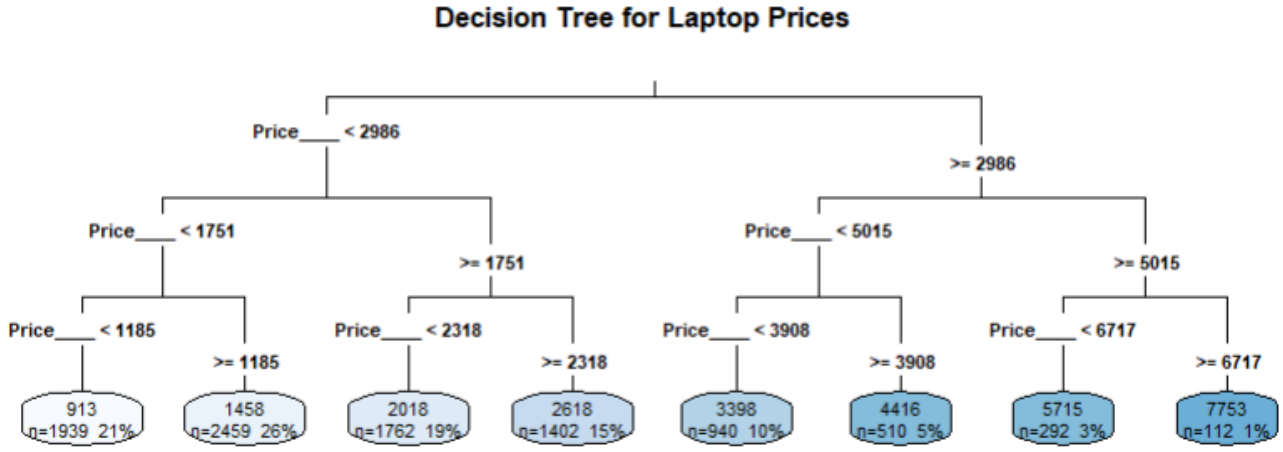
```
data$Total_Pixels <- sapply(resolution_split, function(x) as.numeric(x[1]) * as.numeric(x[2]))
# Remove unnecessary columns
data <- subset(data, select = -c(Resolution))
# Remove missing values
data <- na.omit(data)
# Ensure target variable is numeric
data$Price <- as.numeric(data$Price)
# Split dataset into training and testing (80-20 split)
set.seed(42)
split <- createDataPartition(data$Price, p = 0.8, list = FALSE)
train_data <- data[split, ]
test_data <- data[-split, ]
# Train Decision Tree Model
decision_tree_model <- rpart(Price ~ ., data = train_data, method = "anova")
# Visualize Decision Tree
rpart.plot(decision_tree_model, main = "Decision Tree for Laptop Prices", type = 3, extra = 101)
# Make Predictions
predictions <- predict(decision_tree_model, test_data)
# Evaluate Model Performance
mse <- mean((test_data$Price - predictions)^2)
rmse <- sqrt(mse)
r2 <- 1 - (sum((test_data$Price - predictions)^2) / sum((test_data$Price - mean(test_data$Price))^2))
# Calculate Accuracy (Custom Approach)
tolerance <- 0.1 * test_data$Price  # 10% tolerance
correct_predictions <- abs(test_data$Price - predictions) <= tolerance
accuracy <- mean(correct_predictions) * 100  # Convert to percentage
# Print Final Accuracy Metrics
cat("\n===== Final Model Evaluation Metrics =====\n")
cat(sprintf(" Mean Squared Error (MSE): %.2f\n", mse))
cat(sprintf(" Root Mean Squared Error (RMSE): %.2f\n", rmse))
```

cat(sprintf(" R-squared (R²): %.4f\n", r2))

cat(sprintf(" Model Accuracy: %.2f%%\n", accuracy))

Output:

```
===== Final Model Evaluation Metrics =====
> cat(sprintf("📌 Mean Squared Error (MSE): %.2f\n", mse))
📌 Mean Squared Error (MSE): 54260.56
> cat(sprintf("📌 Root Mean Squared Error (RMSE): %.2f\n", rmse))
📌 Root Mean Squared Error (RMSE): 232.94
> cat(sprintf("📌 R-squared (R²): %.4f\n", r2))
📌 R-squared (R²): 0.9688
> cat(sprintf("✅ Model Accuracy: %.2f%%\n", accuracy))
✅ Model Accuracy: 63.18%
>
```



Python

Code :

import pandas *as* pd

import numpy *as* np

import matplotlib.pyplot *as* plt

from sklearn.model_selection *import* train_test_split

from sklearn.preprocessing *import* LabelEncoder

from sklearn.tree *import* DecisionTreeRegressor, plot_tree

from sklearn.metrics *import* mean_squared_error, r2_score

 # Load dataset

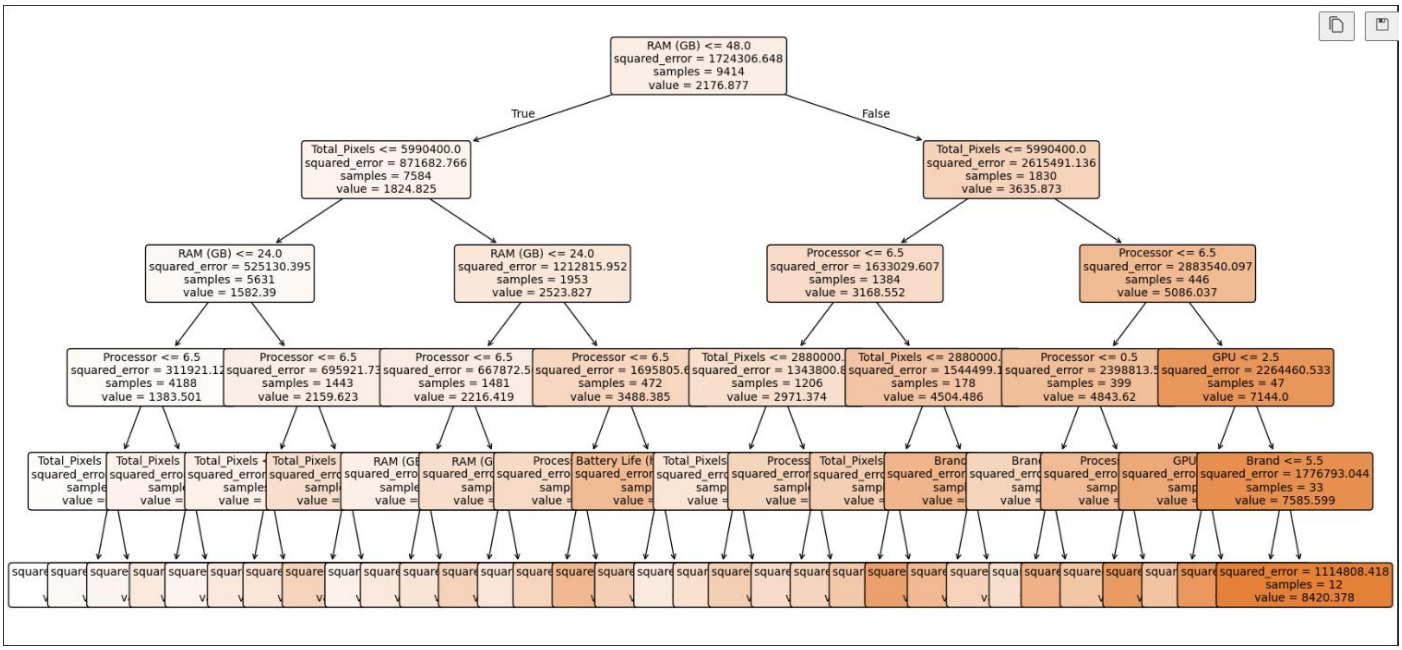
file_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"

```
df = pd.read_csv(file_path)
# Convert Storage to numerical format (handling missing values)
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True
# Encode categorical variables
label_encoders = {}
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
# Define features and target variable
X = df.drop(columns=['Price ($)'])
y = df['Price ($)']
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42, max_depth=5)
dt_model.fit(X_train, y_train)
# Predictions
y_pred = dt_model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
# Calculate accuracy as percentage of predictions within 10% tolerance of actual values
tolerance = 0.1 * y_test  # 10% of actual price
correct_predictions = np.abs(y_test - y_pred) <= tolerance
accuracy = np.mean(correct_predictions) * 100  # Convert to percentage
```

```
# Print evaluation metrics
print(f"MSE: {mse:.2f}")
print(f"R² Score: {r2:.4f}")
print(f"Accuracy: {accuracy:.2f}%")  # Display accuracy percentage
# Visualize the decision tree
plt.figure(figsize=(20, 10))
plot_tree(dt_model, feature_names=X.columns, filled=True, rounded=True, fontsize=10)
plt.show()
```

Output:

```
MSE: 575601.50
R² Score: 0.6752
Accuracy: 24.38%
```



Q2) Navie Bayes

R Programming

Code :

```
# Load required libraries
library(e1071)
library(caret)
library(dplyr)
```

```
# Load dataset
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"
df <- read.csv(file_path, stringsAsFactors = TRUE)
# Check actual column names
print(colnames(df))  # Check column names before processing
# Fix column names (replace spaces and special characters)
colnames(df) <- gsub("[^[:alnum:]_]", "_", colnames(df))
# Identify the actual column name for price
price_col <- grep("Price", colnames(df), value = TRUE)  # Find column with "Price"
if (length(price_col) == 0) {
  stop(" Error: No column related to 'Price' found in the dataset.")
} else {
  cat(" Found Price column:", price_col, "\n")
}
# Convert categorical variables to factors
df$Brand <- as.factor(df$Brand)
df$Processor <- as.factor(df$Processor)
df$GPU <- as.factor(df$GPU)
df$Operating_System <- as.factor(df$Operating_System)
# Convert Storage to numeric (extract digits)
df$Storage <- as.numeric(gsub("[^0-9]", "", df$Storage))
# Convert Resolution to total pixel count
resolution_split <- strsplit(as.character(df$Resolution), "x")
df$Total_Pixels <- sapply(resolution_split, function(x) as.numeric(x[1]) * as.numeric(x[2]))
# Drop unnecessary columns
df <- df %>% select(-Resolution)
# Handle missing values
df <- na.omit(df)
# Convert Price to categorical variable (Low, Medium, High)
df$Price_Category <- cut(df[[price_col]],  # Use dynamic column name
```

```
        breaks = quantile(df[[price_col]], probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),
        labels = c("Low", "Medium", "High"),
        include.lowest = TRUE)
# Remove original Price column
df<- df %>% select(-all_of(price_col))
# Split dataset into training and testing
set.seed(42)
split_index <- createDataPartition(df$Price_Category, p = 0.8, list = FALSE)
train_data <- df[split_index, ]
test_data <- df[-split_index, ]
# Train Naïve Bayes Classifier
nb_model <- naiveBayes(Price_Category ~ ., data = train_data)
# Predictions
predictions <- predict(nb_model, test_data)
# Model Evaluation
accuracy <- mean(predictions == test_data$Price_Category)
cat(sprintf(" Model Accuracy: %.2f%%\n", accuracy * 100))
# Confusion Matrix
cat("\n Confusion Matrix:\n")
print(confusionMatrix(predictions, test_data$Price_Category))
```

Output:

```
Confusion Matrix and Statistics

          Reference
Prediction Low Medium High
    Low    649    230    1
    Medium 131    442  212
    High     4    112  571

Overall Statistics

               Accuracy : 0.7066
                 95% CI : (0.6878, 0.725)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5599

 Mcnemar's Test P-Value : 6.442e-13

Statistics by Class:

                     Class: Low Class: Medium Class: High
Sensitivity              0.8278        0.5638      0.7283
Specificity              0.8527        0.7812      0.9260
Pos Pred Value           0.7375        0.5631      0.8311
Neg Pred Value           0.9083        0.7817      0.8721
Prevalence               0.3333        0.3333      0.3333
Detection Rate           0.2759        0.1879      0.2428
Detection Prevalence     0.3741        0.3338      0.2921
Balanced Accuracy        0.8402        0.6725      0.8272
> |
```

Python

Code :

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load dataset
```

```
file_path = r"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"
df = pd.read_csv(file_path)
# Convert Storage to numerical format (handling missing values)
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)
# Encode categorical variables
label_encoders = {}
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
# Convert price into categories (Low, Medium, High)
df['Price Category'] = pd.qcut(df['Price ($)'], q=3, labels=['Low', 'Medium', 'High'])
# Define features and target variable
X = df.drop(columns=['Price ($)', 'Price Category'])
y = df['Price Category']
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
# Predictions
y_pred = nb_model.predict(X_test)
# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

Output:

```
Accuracy: 0.64

Classification Report:
              precision    recall  f1-score   support

        High       0.78      0.69      0.73       815
         Low       0.63      0.83      0.72       776
      Medium       0.50      0.41      0.45       763

    accuracy                           0.64      2354
   macro avg       0.64      0.64      0.63      2354
weighted avg       0.64      0.64      0.64      2354


Confusion Matrix:
[[561  60 194]
 [ 14 642 120]
 [141 312 310]]
```

Q3) KNN

R Programming

Code :

```
# Load required libraries (install only if missing)
packages <- c("tidyverse", "class", "caret", "ggplot2")
install_if_missing <- function(pkg) {
```

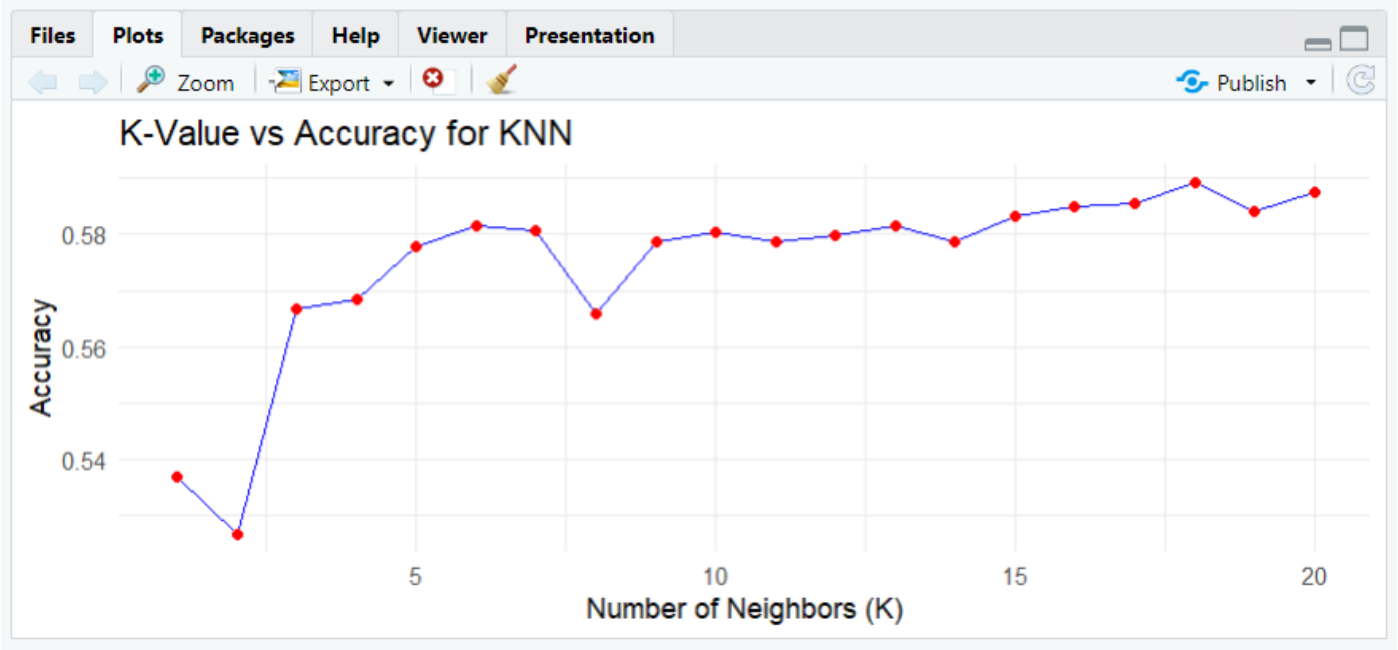
```
  if (!require(pkg, character.only = TRUE)) install.packages(pkg, dependencies = TRUE)
  library(pkg, character.only = TRUE)
}
lapply(packages, install_if_missing)
# Load dataset
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"
df <- read.csv(file_path, stringsAsFactors = FALSE)
# Print column names to verify structure
print("Column names in dataset:")
print(colnames(df))
# Ensure Operating System column exists
actual_os_col <- colnames(df)[grepl("Operating", colnames(df), ignore.case = TRUE)]
if (length(actual_os_col) == 0) {
  stop("Error: Column related to Operating System not found in dataset. Check column names.")
} else {
  colnames(df)[colnames(df) == actual_os_col] <- "Operating_System"
}
# Ensure Price column exists
actual_price_col <- colnames(df)[grepl("Price", colnames(df), ignore.case = TRUE)]
if (length(actual_price_col) == 0) {
  stop("Error: Column related to Price not found in dataset. Check column names.")
} else {
  colnames(df)[colnames(df) == actual_price_col] <- "Price"
}
# Ensure Storage is numeric (remove non-numeric characters)
df$Storage <- as.numeric(gsub("[^0-9]", "", df$Storage))
# Convert Resolution to total pixel count (if column exists)
if ("Resolution" %in% colnames(df)) {
  resolution_split <- strsplit(as.character(df$Resolution), "x")
  df$Total_Pixels <- sapply(resolution_split, function(x) as.numeric(x[1]) * as.numeric(x[2]))
```

```
  df <- df %>% select(-Resolution)  # Remove original Resolution column
}
# Print first few rows to check structure
print(head(df))
# Handle missing values before encoding
df <- df %>% drop_na(Brand, Processor, GPU, Operating_System, Price)
# Encode categorical variables
df$Brand <- as.factor(df$Brand)
df$Processor <- as.factor(df$Processor)
df$GPU <- as.factor(df$GPU)
df$Operating_System <- as.factor(df$Operating_System)
# Convert Price into categories (Low, Medium, High)
df$Price_Category <- cut(df$Price,
                         breaks = quantile(df$Price, probs = seq(0, 1, by = 1/3), na.rm = TRUE),
                         labels = c("Low", "Medium", "High"),
                         include.lowest = TRUE)
# Remove rows with missing values
df <- na.omit(df)
# Define features (X) and target variable (y)
X <- df %>% select(-c(Price, Price_Category))
#  Fix: Select only numeric columns for scaling
X_numeric <- X %>% select(where(is.numeric))
# Check column types
print("Checking feature data types:")
print(str(X_numeric))
# Normalize features
X_scaled <- as.data.frame(scale(X_numeric))
# Split dataset into training and testing sets (80% train, 20% test)
set.seed(42)
train_indices <- createDataPartition(df$Price_Category, p = 0.8, list = FALSE)
```

```
X_train <- X_scaled[train_indices, ]
X_test <- X_scaled[-train_indices, ]
train_labels <- df$Price_Category[train_indices]
test_labels <- df$Price_Category[-train_indices]
# Ensure labels are factors with the same levels
train_labels <- factor(train_labels)
test_labels <- factor(test_labels, levels = levels(train_labels))
# Train KNN model with k=5
k <- 5
knn_pred <- knn(train = X_train, test = X_test, cl = train_labels, k = k)
# Evaluate model performance
accuracy <- mean(knn_pred == test_labels) * 100
cat("KNN Model Accuracy:", round(accuracy, 2), "%\n")
# Confusion matrix
conf_matrix <- confusionMatrix(knn_pred, test_labels)
print(conf_matrix)
# Plot accuracy for different K values
k_values <- seq(1, 20, by = 1)
accuracies <- sapply(k_values, function(k) {
  pred_k <- knn(train = X_train, test = X_test, cl = train_labels, k = k)
  mean(pred_k == test_labels)
})
# Plot the KNN accuracy curve
ggplot(data.frame(K = k_values, Accuracy = accuracies), aes(x = K, y = Accuracy)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  ggtitle("K-Value vs Accuracy for KNN") +
  xlab("Number of Neighbors (K)") +
  ylab("Accuracy") +
  theme_minimal()
```

Output:

```
Confusion Matrix and Statistics

          Reference
Prediction Low Medium High
    Low    514    269   62
    Medium 220    299  203
    High    50    216  519

Overall Statistics

               Accuracy : 0.5663
                 95% CI : (0.546, 0.5865)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.3495

 Mcnemar's Test P-Value : 0.08584

Statistics by Class:

                     Class: Low Class: Medium Class: High
Sensitivity              0.6556        0.3814      0.6620
Specificity              0.7889        0.7302      0.8304
Pos Pred Value           0.6083        0.4141      0.6611
Neg Pred Value           0.8208        0.7025      0.8309
Prevalence               0.3333        0.3333      0.3333
Detection Rate           0.2185        0.1271      0.2207
Detection Prevalence     0.3593        0.3070      0.3338
Balanced Accuracy        0.7223        0.5558      0.7462
>
```



Python

Code :

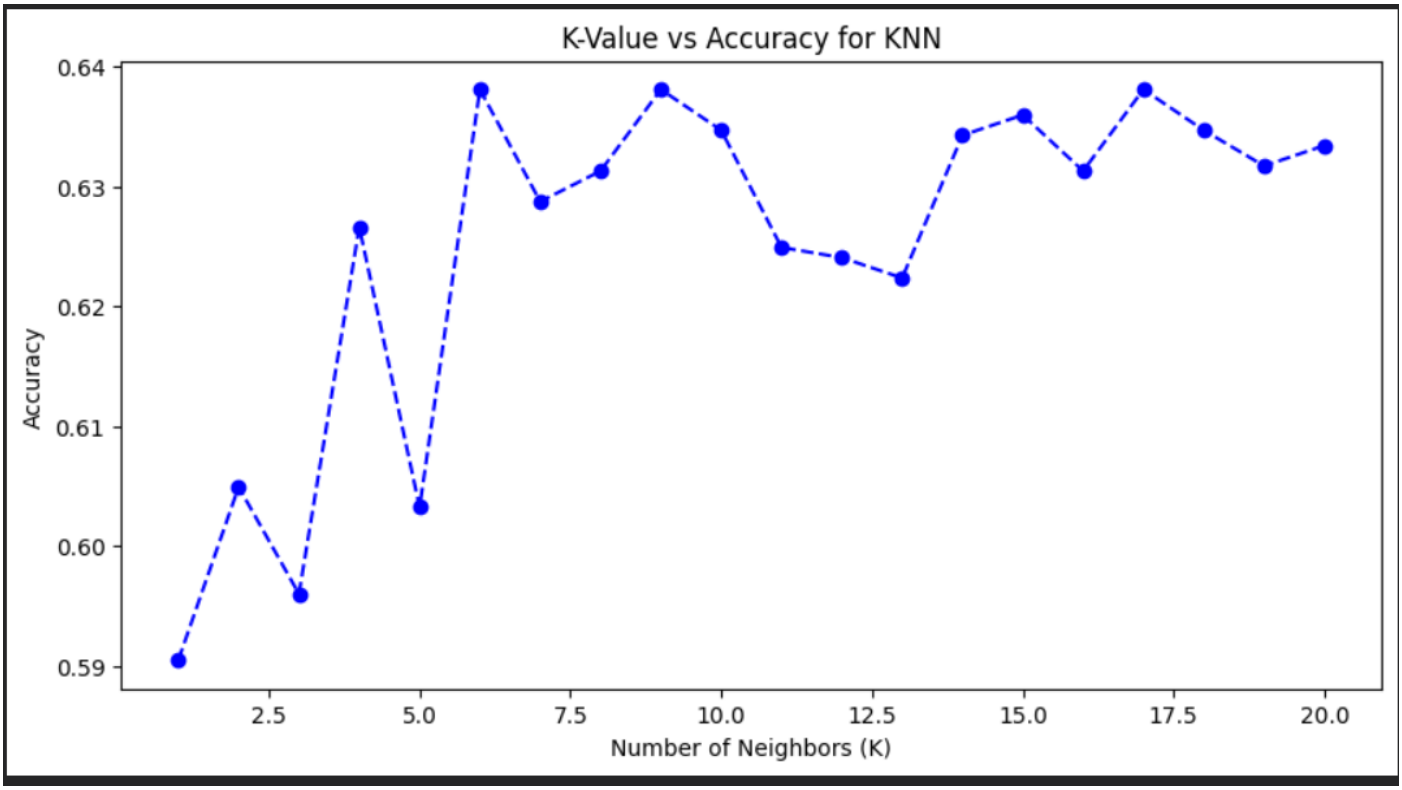
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Load dataset
file_path = r"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"
df = pd.read_csv(file_path)
# Convert Storage to numerical format (handling missing values)
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)
# Encode categorical variables
label_encoders = {}
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
# Convert price into categories (Low, Medium, High)
df['Price Category'] = pd.qcut(df['Price ($)'], q=3, labels=['Low', 'Medium', 'High'])
# Define features and target variable
X = df.drop(columns=['Price ($)', 'Price Category'])
y = df['Price Category']
```

```
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train KNN Classifier
k = 5  # Number of neighbors
knn_model = KNeighborsClassifier(n_neighbors=k)
knn_model.fit(X_train, y_train)
# Predictions
y_pred = knn_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
# Plot accuracy for different K values
k_values = range(1, 21)
accuracies = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred_k))
plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracies, marker='o', linestyle='dashed', color='b')
```

```
plt.xlabel("Number of Neighbors (K)")
plt.ylabel("Accuracy")
plt.title("K-Value vs Accuracy for KNN")
plt.show()
```

Output:

```
Accuracy: 0.60

Classification Report:
              precision    recall  f1-score   support

        High       0.68      0.72      0.70       815
         Low       0.64      0.74      0.69       776
      Medium       0.44      0.33      0.38       763

    accuracy                           0.60      2354
   macro avg       0.58      0.60      0.59      2354
weighted avg       0.59      0.60      0.59      2354


Confusion Matrix:
[[590  55 170]
 [ 47 577 152]
 [234 276 253]]
```



Q4) SVM

R Programming

Code :

```
# Load required libraries
if (!require("tidyverse")) install.packages("tidyverse", dependencies = TRUE)
if (!require("e1071")) install.packages("e1071")  # SVM
if (!require("caret")) install.packages("caret")  # Data Processing
library(tidyverse)
library(e1071)
library(caret)
# Load dataset
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"
df <- read.csv(file_path, stringsAsFactors = FALSE)
#  Step 1: Fix Column Names
colnames(df) <- gsub("\\.", "_", colnames(df))  # Replace dots with underscores
colnames(df) <- gsub(" ", "_", colnames(df))    # Replace spaces with underscores
#  Step 2: Identify the Correct 'Price' Column
price_col <- grep("Price", colnames(df), value = TRUE)
if (length(price_col) == 0) stop("Error: 'Price' column not found in dataset!")
cat("Found Price Column:", price_col, "\n")
colnames(df)[colnames(df) == price_col] <- "Price"  # Rename to 'Price'
#  Step 3: Convert Storage to Numeric
df$Storage <- as.numeric(gsub("[^0-9]", "", df$Storage))
#  Step 4: Convert Resolution to Total Pixels
if ("Resolution" %in% colnames(df)) {
  resolution_split <- strsplit(as.character(df$Resolution), "x")
  df$Total_Pixels <- sapply(resolution_split, function(x) as.numeric(x[1]) * as.numeric(x[2]))
  df <- df %>% select(-Resolution)  # Remove Resolution column
}
#  Step 5: Handle Missing Values
df <- df %>% drop_na(Brand, Processor, GPU, Operating_System, Price)
```

```
#  Step 6: Encode Categorical Variables as Factors
df$Brand <- as.factor(df$Brand)
df$Processor <- as.factor(df$Processor)
df$GPU <- as.factor(df$GPU)
df$Operating_System <- as.factor(df$Operating_System)
#  Step 7: Convert Price into Categories (Low, Medium, High)
df$Price_Category <- cut(df$Price,
                         breaks = quantile(df$Price, probs = seq(0, 1, by = 1/3), na.rm = TRUE),
                         labels = c("Low", "Medium", "High"),
                         include.lowest = TRUE)
#  Step 8: Remove NA Rows
df <- na.omit(df)
#  Step 9: Convert Categorical Variables to Numeric using One-Hot Encoding
df_numeric <- model.matrix(~ . - 1, data = df %>% select(-Price_Category)) %>% as.data.frame()
#  Step 10: Define Features (X) and Target Variable (y)
X <- df_numeric
y <- df$Price_Category
#  Step 11: Normalize Features (Fixing colMeans error)
X_scaled <- scale(X)  # Ensures all columns are numeric before scaling
#  Step 12: Split Data (80-20)
set.seed(42)
train_indices <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X_scaled[train_indices, ]
X_test <- X_scaled[-train_indices, ]
train_labels <- y[train_indices]
test_labels <- y[-train_indices]
#  Step 13: Train SVM Model
svm_model <- svm(train_labels ~ ., data = X_train, kernel = "linear", cost = 1)
#  Step 14: Make Predictions
svm_pred <- predict(svm_model, X_test)
```

```
# Step 15: Evaluate Model Performance
accuracy <- mean(svm_pred == test_labels) * 100
cat("SVM Model Accuracy:", round(accuracy, 2), "%\n")
# Step 16: Confusion Matrix
conf_matrix <- confusionMatrix(as.factor(svm_pred), as.factor(test_labels))
print(conf_matrix)
```

Output:

```
SVM Model Accuracy: 98.85 %
>
> #  🔹 Step 16: Confusion Matrix
> conf_matrix <- confusionMatrix(as.factor(svm_pred), as.factor(test_labels))
> print(conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction Low Medium High
    Low    777      8    0
    Medium   7    771    7
    High     0      5  777

Overall Statistics

               Accuracy : 0.9885
                 95% CI : (0.9833, 0.9924)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9828

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Low Class: Medium Class: High
Sensitivity              0.9911        0.9834      0.9911
Specificity              0.9949        0.9911      0.9968
Pos Pred Value           0.9898        0.9822      0.9936
Neg Pred Value           0.9955        0.9917      0.9955
Prevalence               0.3333        0.3333      0.3333
Detection Rate           0.3304        0.3278      0.3304
Detection Prevalence     0.3338        0.3338      0.3325
Balanced Accuracy        0.9930        0.9872      0.9939
>
```

Python

Code :

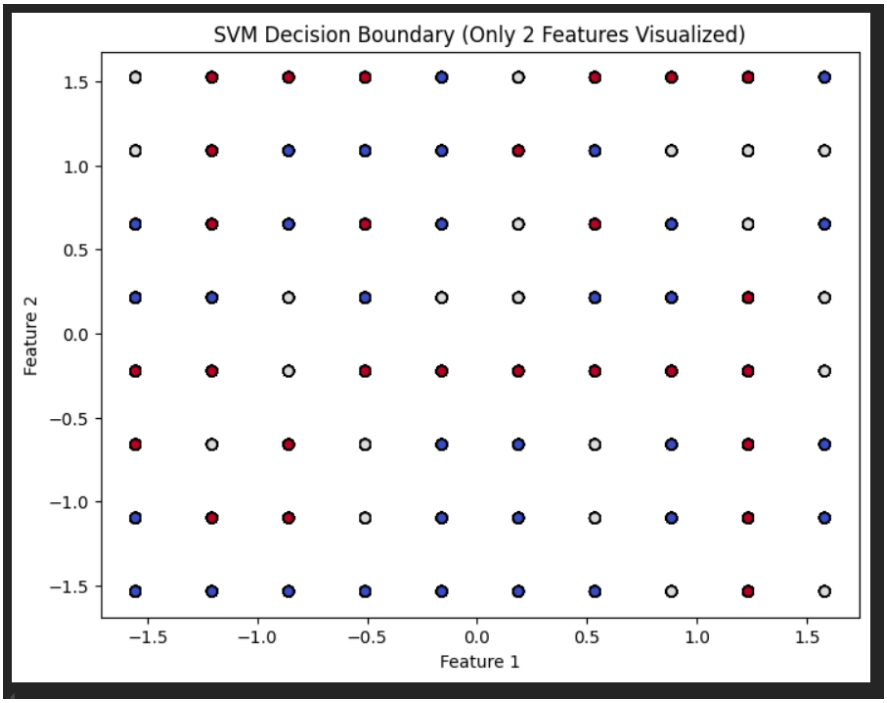
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Load dataset
file_path = r"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"
df = pd.read_csv(file_path)
# Convert Storage to numerical format (handling missing values)
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)
# Encode categorical variables
label_encoders = {}
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
# Convert price into categories (Low, Medium, High)
df['Price Category'] = pd.qcut(df['Price ($)'], q=3, labels=['Low', 'Medium', 'High'])
# Define features and target variable
X = df.drop(columns=['Price ($)', 'Price Category'])
y = df['Price Category']
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train Support Vector Machine (SVM) Classifier
```

```
svm_model = SVC(kernel='linear', C=1.0)  # Linear Kernel
svm_model.fit(X_train, y_train)
# Predictions
y_pred = svm_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
# Visualization (Only for 2D data, choosing first two features)
plt.figure(figsize=(8, 6))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train.astype('category').cat.codes, cmap='coolwarm', edgecolors='k')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("SVM Decision Boundary (Only 2 Features Visualized)")
plt.show()
```

Output :

```
Accuracy: 0.67

Classification Report:
              precision    recall  f1-score   support

        High       0.74      0.78      0.76       815
         Low       0.71      0.79      0.75       776
      Medium       0.53      0.44      0.48       763

    accuracy                           0.67      2354
   macro avg       0.66      0.67      0.66      2354
weighted avg       0.66      0.67      0.66      2354


Confusion Matrix:
[[632  29 154]
 [ 20 612 144]
 [202 224 337]]
```



Q5) Kmeans

R Programming

Code :

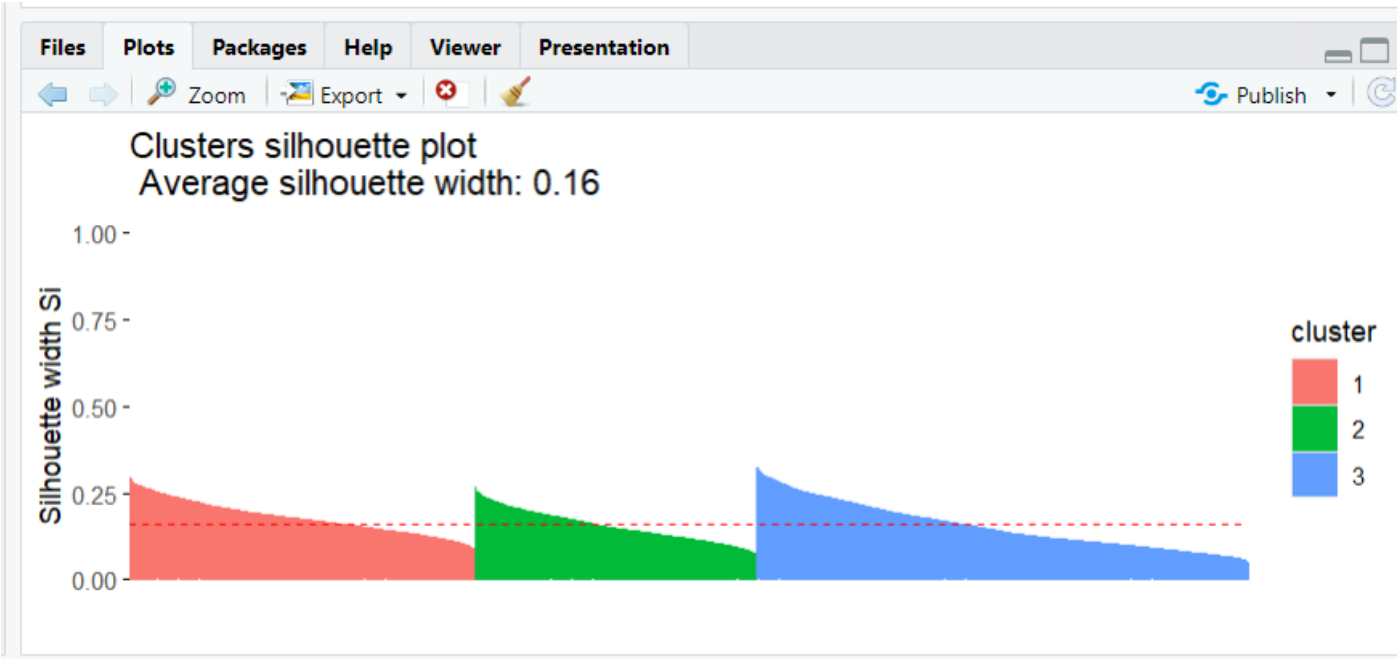
```
# Load required libraries
library(dplyr)
library(ggplot2)
library(cluster)
library(factoextra)
library(NbClust)
```

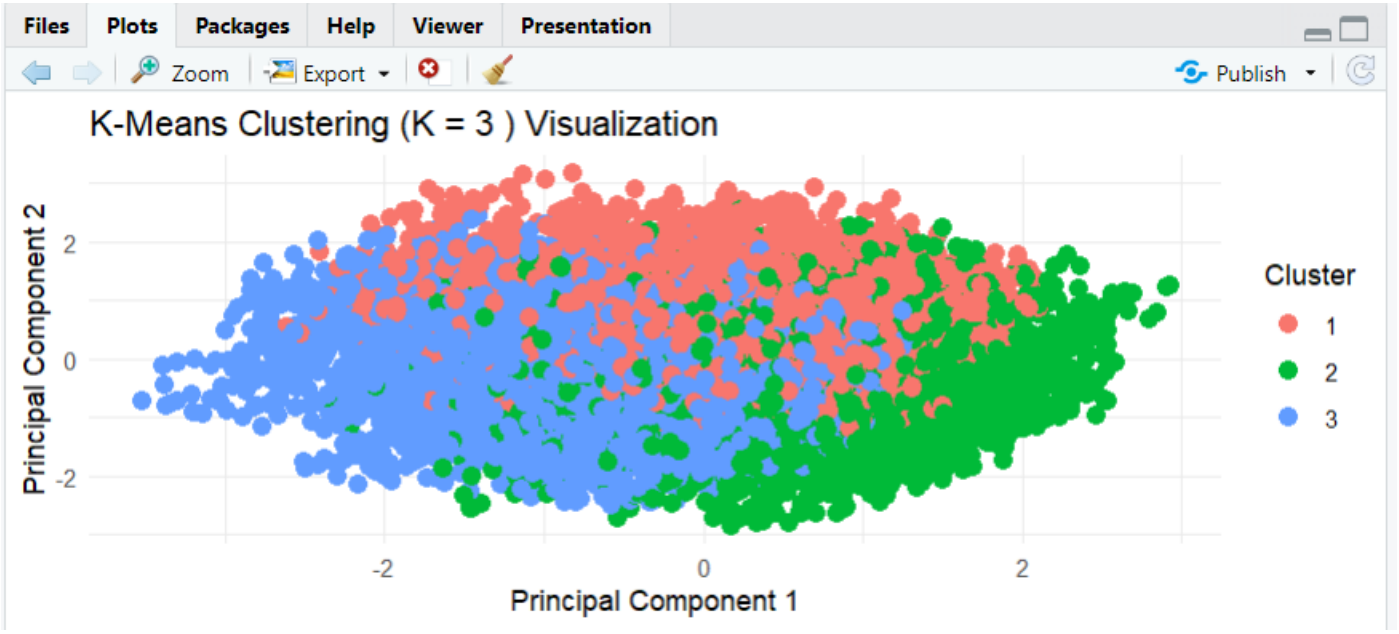
```
library(readr)
library(tidyr)
# Load dataset
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"
df <- read_csv(file_path)
# Print column names for reference
print(colnames(df))
# Detect the "Price" column dynamically
price_col <- grep("price", tolower(gsub("[^a-zA-Z0-9]", "", colnames(df))), value = TRUE)
# Ensure the Price column exists
if (length(price_col) == 0) {
  stop(" Error: 'Price' column not found in dataset! Check column names above.")
} else {
  cat(" Found Price column:", price_col, "\n")
}
# Drop Price column
df <- df %>% select(-matches(price_col))
# Convert Storage to numeric (extract numbers safely)
if ("Storage" %in% colnames(df)) {
  df <- df %>%
    mutate(Storage = as.numeric(gsub("[^0-9]", "", Storage)))
}
# Convert Resolution to pixel count (handle missing values)
if ("Resolution" %in% colnames(df)) {
  df <- df %>%
    separate(Resolution, into = c("Width", "Height"), sep = "x", convert = TRUE, fill = "right") %>%
    mutate(Total_Pixels = as.numeric(Width) * as.numeric(Height)) %>%
    select(-Width, -Height)
}
# Identify categorical columns dynamically
```

```
categorical_cols <- c("Brand", "Processor", "GPU", "Operating System")
categorical_cols <- categorical_cols[categorical_cols %in% colnames(df)]
# Convert categorical columns to factors
if (length(categorical_cols) > 0) {
  df[categorical_cols] <- lapply(df[categorical_cols], as.factor)
}
# Handle missing values
df <- df %>%
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), median(., na.rm = TRUE), .)))
# Convert NaN and Inf values to NA, then replace with median
df <- df %>%
  mutate(across(where(is.numeric), ~ ifelse(is.nan(.), NA, .))) %>%
  mutate(across(where(is.numeric), ~ ifelse(is.infinite(.), NA, .)))
# Final missing value check
df <- df %>%
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), median(., na.rm = TRUE), .)))
# Ensure no NA/NaN/Inf remains
if (any(is.na(df))) stop(" Error: NA values still exist in the dataset after preprocessing!")
if (any(is.nan(as.matrix(df %>% select(where(is.numeric)))))) stop(" Error: NaN values detected!")
if (any(!is.finite(as.matrix(df %>% select(where(is.numeric)))))) stop(" Error: Inf values detected!")
# Remove categorical columns before clustering
df <- df %>% select(where(is.numeric))
# Standardize numeric features
df <- scale(df)
# Find optimal K using the Elbow Method
wss <- sapply(1:10, function(k) {
  kmeans(df, centers = k, nstart = 10)$tot.withinss
})
# Plot Elbow Method graph
plot(1:10, wss, type = "b", pch = 19, col = "blue",
```

```
    xlab = "Number of Clusters (K)", ylab = "Total Within Sum of Squares",
    main = "Elbow Method for Optimal K")
# Choose optimal K
optimal_k <- 3
# Apply K-Means clustering
set.seed(42)
kmeans_model <- kmeans(df, centers = optimal_k, nstart = 10)
# Add cluster labels
df_clustered <- as.data.frame(df)
df_clustered$Cluster <- as.factor(kmeans_model$cluster)
# PCA for visualization
pca_result <- prcomp(df, center = TRUE, scale. = TRUE)
df_pca <- as.data.frame(pca_result$x[, 1:2])
df_pca$Cluster <- df_clustered$Cluster
# Scatter plot of clusters
ggplot(df_pca, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(size = 3) +
  labs(title = paste("K-Means Clustering (K =", optimal_k, ") Visualization"),
       x = "Principal Component 1", y = "Principal Component 2") +
  theme_minimal()
```

Output:






```
  cluster size ave.sil.width
1       1 3623          0.18
2       2 2977          0.15
3       3 5168          0.15
>
> # Average Silhouette Score
> avg_silhouette <- mean(silhouette_score[, 3])
> cat("✅ Average Silhouette Score:", avg_silhouette, "\n")
✅ Average Silhouette Score: 0.1615929
>
```

Python

Code :

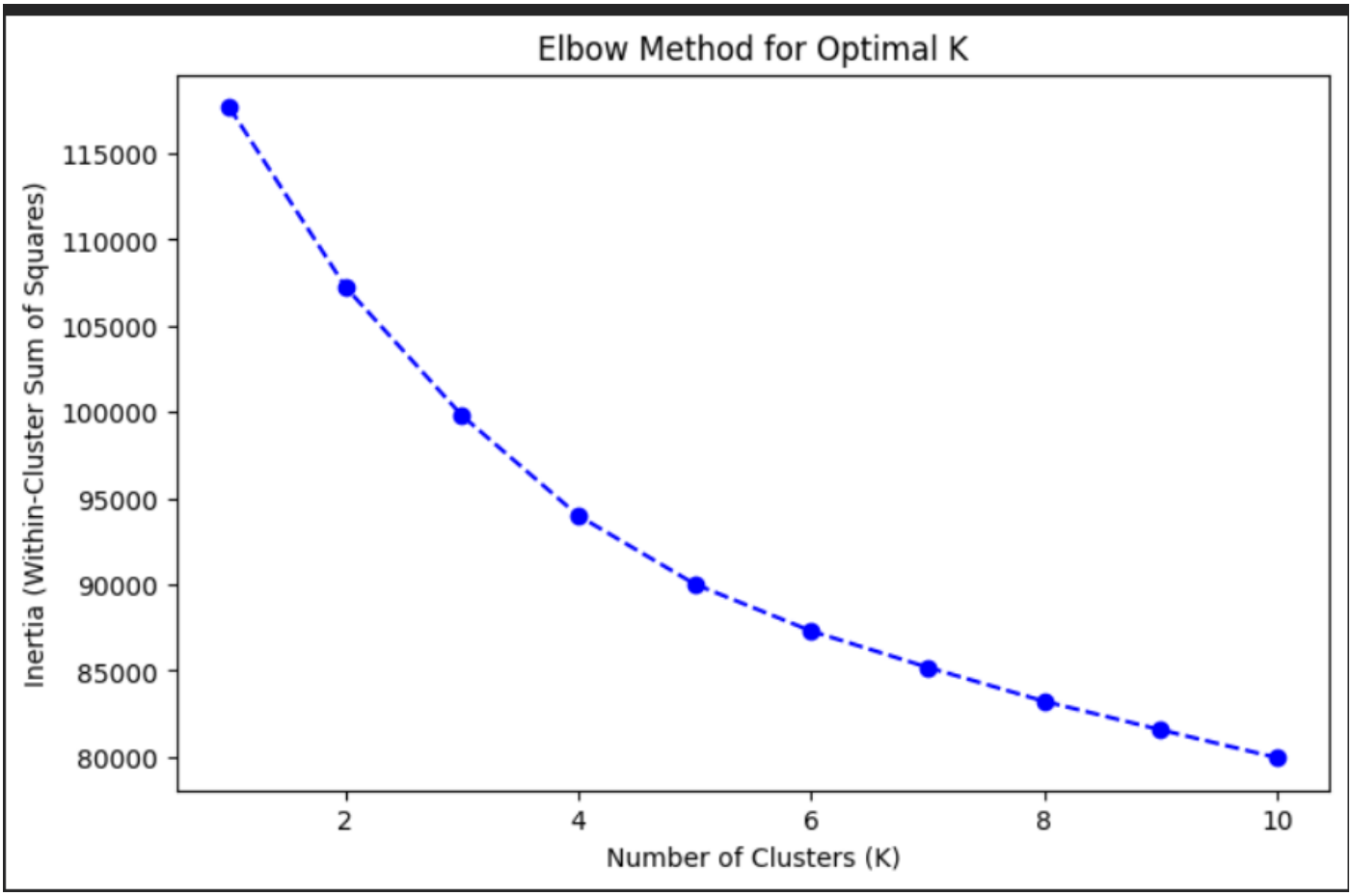
import pandas *as* pd

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
from sklearn.utils import resample
# Load dataset
file_path = r"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"
df = pd.read_csv(file_path)
# Convert Storage to numerical format (handling missing values)
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)
# Encode categorical variables
label_encoders = {}
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
# Define features for clustering (excluding price)
X = df.drop(columns=['Price ($)'])
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Find optimal K using Elbow Method
inertia = []
```

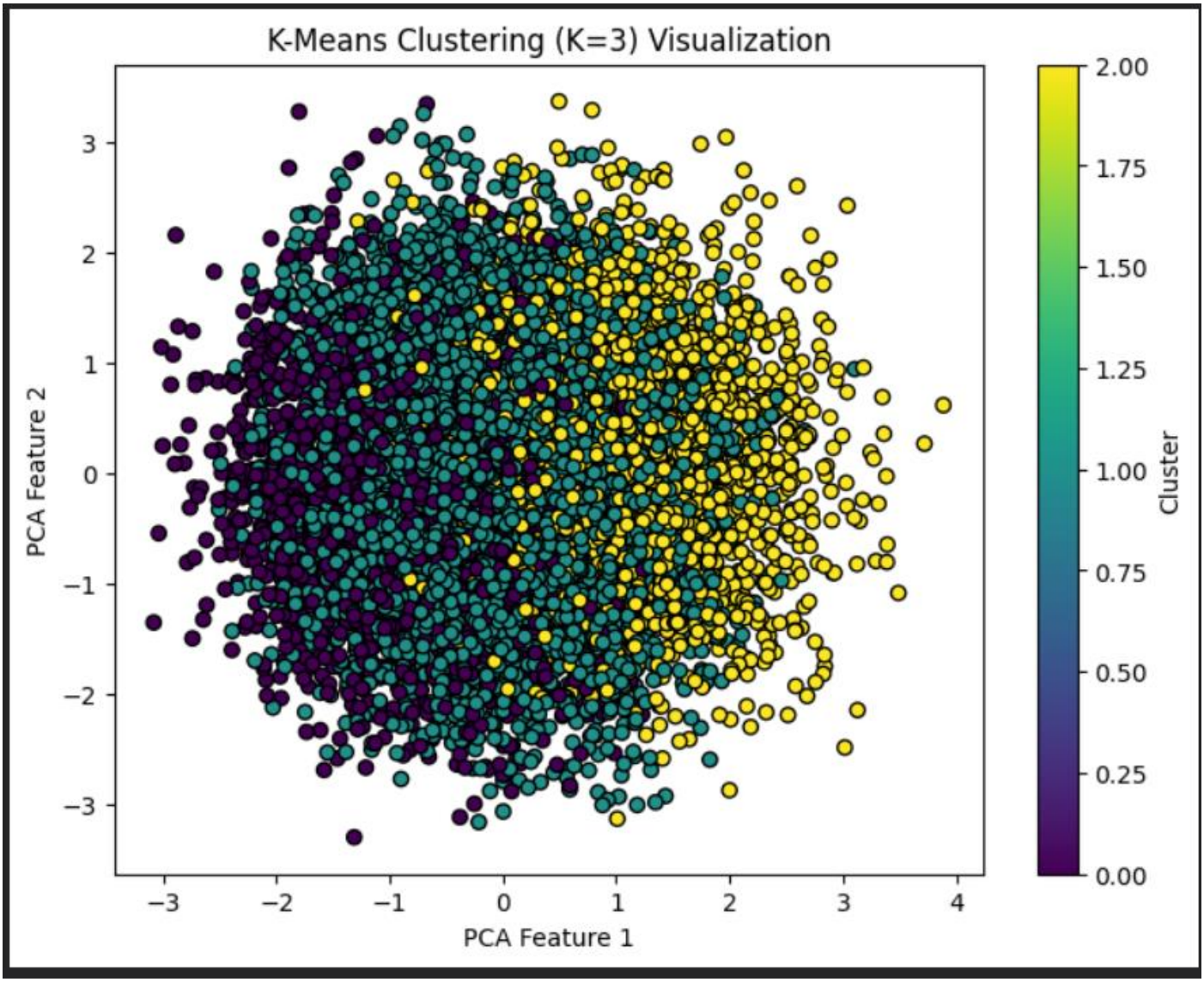
```
K_range = range(1, 11)  # Checking K values from 1 to 10
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
# Plot Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o', linestyle='--', color='b')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia (Within-Cluster Sum of Squares)")
plt.title("Elbow Method for Optimal K")
plt.show()
# Train K-Means model with optimal K (choose K based on elbow graph, e.g., 3)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)
# Take a random sample to avoid MemoryError
sample_size = min(10000, len(X_scaled))  # Ensure it doesn't exceed dataset size
X_sample, labels_sample = resample(X_scaled, df['Cluster'], n_samples=sample_size, random_state=42)
# Evaluate Clustering Performance
silhouette_avg = silhouette_score(X_sample, labels_sample)
davies_bouldin = davies_bouldin_score(X_sample, labels_sample)
calinski_harabasz = calinski_harabasz_score(X_sample, labels_sample)
# Print evaluation metrics
print("===== Final Clustering Metrics =====")
print(f"Silhouette Score: {silhouette_avg:.4f} (Higher is better)")
print(f"Davies-Bouldin Index: {davies_bouldin:.4f} (Lower is better)")
print(f"Calinski-Harabasz Index: {calinski_harabasz:.4f} (Higher is better)")
# Reduce dimensionality for visualization using PCA
pca = PCA(n_components=2)
```

```
X_pca = pca.fit_transform(X_scaled)
# Scatter plot of clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['Cluster'], cmap='viridis', edgecolors='k')
plt.xlabel("PCA Feature 1")
plt.ylabel("PCA Feature 2")
plt.title(f"K-Means Clustering (K={optimal_k}) Visualization")
plt.colorbar(label="Cluster")
plt.show()
```

Output:



```
===== Final Clustering Metrics =====
Silhouette Score: 0.1087 (Higher is better)
Davies-Bouldin Index: 2.5912 (Lower is better)
Calinski-Harabasz Index: 898.8993 (Higher is better)
```



Q6) RandomForest

Weka

Output:

R Programming

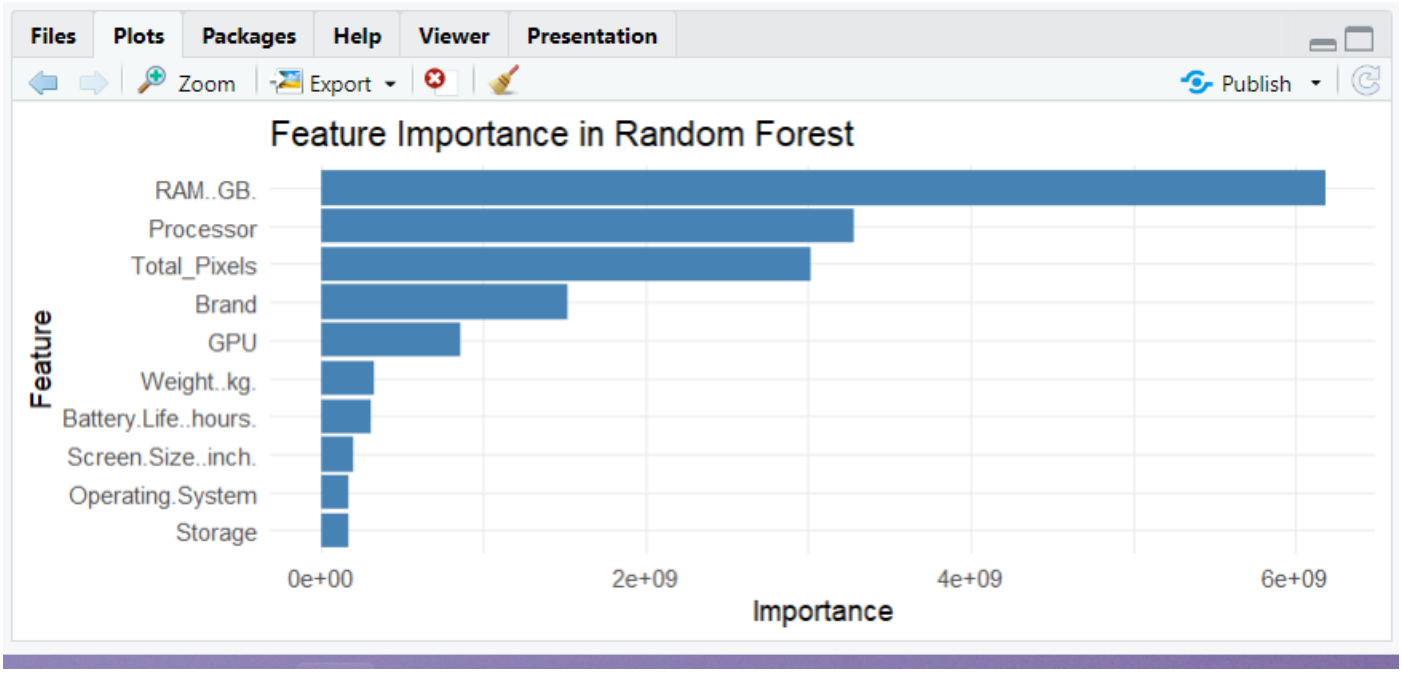
Code :

```
# Load necessary libraries

if (!require(randomForest)) install.packages("randomForest", dependencies=TRUE)
```

```
if (!require(ggplot2)) install.packages("ggplot2", dependencies=TRUE)
if (!require(caret)) install.packages("caret", dependencies=TRUE)
if (!require(dplyr)) install.packages("dplyr", dependencies=TRUE)
library(randomForest)
library(ggplot2)
library(caret)
library(dplyr)
# Load dataset
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"
df <- read.csv(file_path, stringsAsFactors = FALSE)
# Convert Storage to numerical format (handling missing values)
df$Storage <- as.numeric(gsub("\\D", "", df$Storage))
# Convert Resolution to total pixel count
resolution_split <- strsplit(df$Resolution, "x")
df$Width <- as.numeric(sapply(resolution_split, `[`, 1))
df$Height <- as.numeric(sapply(resolution_split, `[`, 2))
df$Total_Pixels <- df$Width * df$Height
df <- df %>% select(-Resolution, -Width, -Height)
# Encode categorical variables
categorical_cols <- c("Brand", "Processor", "GPU", "Operating.System")
df[categorical_cols] <- lapply(df[categorical_cols], as.factor)
# Define features and target variable
target <- "Price...."
X <- df %>% select(-all_of(target))
y <- df[[target]]
# Split dataset into training and testing sets
set.seed(42)
train_indices <- createDataPartition(y, p = 0.8, list = FALSE)
train_data <- df[train_indices, ]
test_data <- df[-train_indices, ]
```

```
# Train Random Forest Regressor
rf_model <- randomForest(Price.... ~ ., data = train_data, ntree = 100, importance = TRUE, seed = 42)


# Predictions
rf_pred <- predict(rf_model, test_data)
# Evaluate the model
mae <- mean(abs(test_data$Price.... - rf_pred))
mse <- mean((test_data$Price.... - rf_pred)^2)
rmse <- sqrt(mse)
r2 <- cor(test_data$Price...., rf_pred)^2
# Compute Adjusted R²
n <- nrow(test_data)  # Number of test samples
p <- ncol(test_data) - 1  # Number of features
adjusted_r2 <- 1 - (1 - r2) * (n - 1) / (n - p - 1)
# Print accuracy metrics
data.frame(
  Metric = c("Mean Absolute Error", "Mean Squared Error", "Root Mean Squared Error", "R² Score", "Adjusted R² Score", "Final Accuracy (R² Score)"),
  Value = c(round(mae, 2), round(mse, 2), round(rmse, 2), round(r2, 4), round(adjusted_r2, 4), round(r2 * 100, 2))
)
# Feature Importance Visualization
importance_df <- as.data.frame(importance(rf_model))
importance_df$Feature <- rownames(importance_df)
importance_df <- importance_df %>% arrange(desc(IncNodePurity))
ggplot(importance_df, aes(x = reorder(Feature, IncNodePurity), y = IncNodePurity)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Feature Importance in Random Forest", x = "Feature", y = "Importance") +
  theme_minimal()
```

Output:



	Metric	Value
1	Mean Absolute Error	184.6900
2	Mean Squared Error	72099.9900
3	Root Mean Squared Error	268.5100
4	R² Score	0.9708
5	Adjusted R² Score	0.9707
6	Final Accuracy (R² Score)	97.0800

Python

Code :

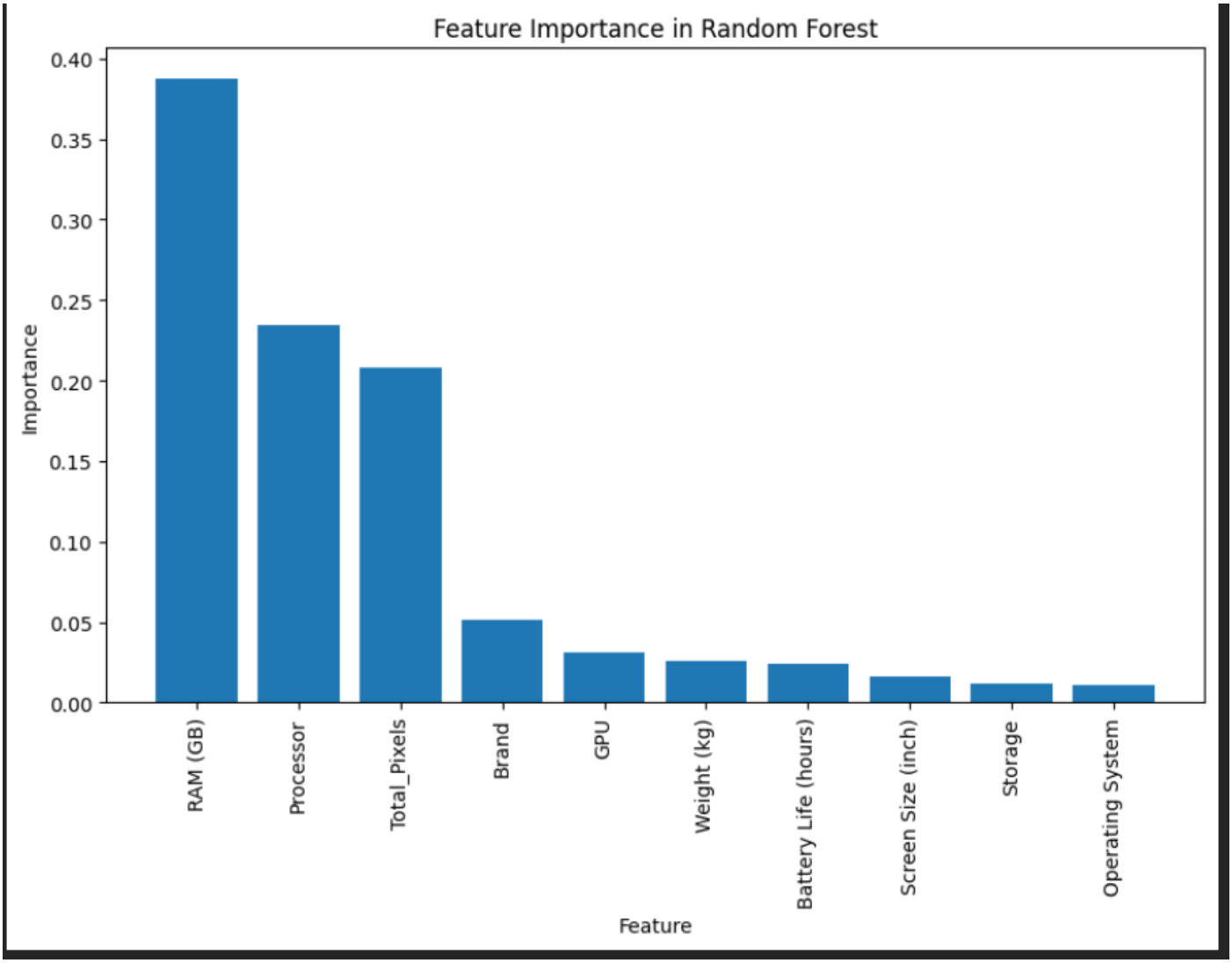
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Load dataset
file_path = r"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"
df = pd.read_csv(file_path)
# Convert Storage to numerical format (handling missing values)
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
```

```
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)
# Encode categorical variables
label_encoders = {}
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
# Define features and target variable
X = df.drop(columns=['Price ($)'])
y = df['Price ($)']
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Predictions
y_pred = rf_model.predict(X_test)
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
# Compute Adjusted R²
n = X_test.shape[0]  # Number of test samples
p = X_test.shape[1]  # Number of features
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
```

```
# Print accuracy metrics
print("=== Model Performance ===")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.4f}")
print(f"Adjusted R² Score: {adjusted_r2:.4f}")
print(f"Final Accuracy (R² Score): {r2 * 100:.2f}%")
# Feature Importance Visualization
feature_importances = rf_model.feature_importances_
sorted_indices = np.argsort(feature_importances)[::-1]
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_importances[sorted_indices], align="center")
plt.xticks(range(X.shape[1]), X.columns[sorted_indices], rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature Importance in Random Forest")
plt.show()
```

Output:

```
=== Model Performance ===
Mean Absolute Error (MAE): 313.77
Mean Squared Error (MSE): 213056.90
Root Mean Squared Error (RMSE): 461.58
R² Score: 0.8798
Adjusted R² Score: 0.8793
Final Accuracy (R² Score): 87.98%
```



Q7) XdBoost Algoritham

R Programming

Code :

```
library(xgboost)
library(caret)
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"
df <- read.csv(file_path)
df$Storage <- as.numeric(gsub("\\D", "", df$Storage))
resolution_split <- strsplit(df$Resolution, "x")
df$Width <- as.numeric(sapply(resolution_split, `[`, 1))
df$Height <- as.numeric(sapply(resolution_split, `[`, 2))
df$Total_Pixels <- df$Width * df$Height
```

```
df <- df[ , !(names(df) %in% c("Resolution", "Width", "Height"))]
df <- na.omit(df)
df$Brand <- as.numeric(factor(df$Brand))
df$Processor <- as.numeric(factor(df$Processor))
df$GPU <- as.numeric(factor(df$GPU))
df$`Operating.System` <- as.numeric(factor(df$`Operating.System`))
X <- df[, !names(df) %in% c("Price..")]
y <- df$Price..
set.seed(42)
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[train_index, ]
y_train <- y[train_index]
X_test <- X[-train_index, ]
y_test <- y[-train_index]
dtrain <- xgb.DMatrix(data = as.matrix(X_train), label = y_train)
dtest <- xgb.DMatrix(data = as.matrix(X_test), label = y_test)
param <- list(
  objective = "reg:squarederror",
  eta = 0.1,
  max_depth = 6,
  nrounds = 100,
  subsample = 0.8,
  colsample_bytree = 0.8
)
xgb_model <- xgb.train(params = param, data = dtrain, nrounds = param$nrounds)
predictions <- predict(xgb_model, newdata = dtest)
mse <- mean((y_test - predictions)^2)
r2 <- 1 - sum((y_test - predictions)^2) / sum((y_test - mean(y_test))^2)


cat("MSE:", mse, "\nR² Score:", r2, "\n")
```

cat("Final Accuracy (R² Score):", r2 * 100, "%\n")

Output:

```
> cat("MSE: ", mse, "\nR² Score: ", r2, "\n")
MSE: 315.0649
R² Score: 0.9998188
> cat("Final Accuracy (R² Score):", r2 * 100, "%\n")
Final Accuracy (R² Score): 99.98188 %
> *
```

Python

Code :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Load dataset
file_path = r"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"
df = pd.read_csv(file_path)
# Convert Storage to numerical format (handling missing values)
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)
# Encode categorical variables
label_encoders = {}
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    le = LabelEncoder()
```

```
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
# Define features and target variable
X = df.drop(columns=['Price ($)'])
y = df['Price ($)']
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Fix: Use 'estimator' instead of 'base_estimator'
adaboost_model = AdaBoostRegressor(
    estimator=DecisionTreeRegressor(max_depth=5),  # Fixed
    n_estimators=50,
    random_state=42
)
# Train the model
adaboost_model.fit(X_train, y_train)
# Predictions
y_pred = adaboost_model.predict(X_test)
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
# Compute Adjusted R² Score
n = X_test.shape[0]  # Number of test samples
p = X_test.shape[1]  # Number of features
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
# Final Accuracy Results
print("=== Model Performance ===")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

```
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.4f}")
print(f"Adjusted R² Score: {adjusted_r2:.4f}")
print(f"Final Accuracy (R² Score): {r2 * 100:.2f}%")
# Plot feature importance
feature_importances = adaboost_model.feature_importances_
sorted_indices = np.argsort(feature_importances)[::-1]
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_importances[sorted_indices], align="center")
plt.xticks(range(X.shape[1]), X.columns[sorted_indices], rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature Importance in AdaBoost")
plt.show()
```

Output:

```
=== Model Performance ===
Mean Absolute Error (MAE): 544.49
Mean Squared Error (MSE): 399930.70
Root Mean Squared Error (RMSE): 632.40
R² Score: 0.7743
Adjusted R² Score: 0.7734
Final Accuracy (R² Score): 77.43%
```

Q8) Stacking

R Programming

Code :

```
library(caret)
library(randomForest)
library(xgboost)
library(e1071)
file_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop_prices.csv"
df <- read.csv(file_path)
df$Storage <- as.numeric(gsub("\\D", "", df$Storage))
resolution_split <- strsplit(df$Resolution, "x")
df$Width <- as.numeric(sapply(resolution_split, `[`, 1))
df$Height <- as.numeric(sapply(resolution_split, `[`, 2))
df$Total_Pixels <- df$Width * df$Height
df <- df[ , !(names(df) %in% c("Resolution", "Width", "Height"))]
df <- na.omit(df)
df$Brand <- as.numeric(factor(df$Brand))
df$Processor <- as.numeric(factor(df$Processor))
df$GPU <- as.numeric(factor(df$GPU))
df$`Operating.System` <- as.numeric(factor(df$`Operating.System`))
X <- df[, !names(df) %in% c("Price..")]
y <- df$Price..
set.seed(42)
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[train_index, ]
y_train <- y[train_index]
X_test <- X[-train_index, ]
y_test <- y[-train_index]
decision_tree <- train(y_train ~ ., data = cbind(X_train, y_train), method = "rpart", tuneLength = 10)
random_forest <- randomForest(x = X_train, y = y_train, ntree = 10)
dtrain <- xgb.DMatrix(data = as.matrix(X_train), label = y_train)
dtest <- xgb.DMatrix(data = as.matrix(X_test), label = y_test)
param <- list(
```

```
  objective = "reg:squarederror",
  eta = 0.1,
  max_depth = 6,
  nrounds = 100,
  subsample = 0.8,
  colsample_bytree = 0.8
)
xgb_model <- xgb.train(params = param, data = dtrain, nrounds = param$nrounds)
meta_train_data <- data.frame(
  DT_pred = predict(decision_tree, X_train),
  RF_pred = predict(random_forest, X_train),
  XGB_pred = predict(xgb_model, newdata = as.matrix(X_train)),
  y_train = y_train
)
meta_model <- lm(y_train ~ ., data = meta_train_data)
predict_dt <- predict(decision_tree, X_test)
predict_rf <- predict(random_forest, X_test)
predict_xgb <- predict(xgb_model, newdata = as.matrix(X_test))
meta_test_data <- data.frame(
  DT_pred = predict_dt,
  RF_pred = predict_rf,
  XGB_pred = predict_xgb
)
meta_pred <- predict(meta_model, newdata = meta_test_data)
mse <- mean((y_test - meta_pred)^2)
r2 <- 1 - sum((y_test - meta_pred)^2) / sum((y_test - mean(y_test))^2)
cat("MSE:", mse, "\nR² Score:", r2, "\n")
cat("Final Accuracy (R² Score):", r2 * 100, "%\n")
```

Output:

```
> cat("MSE:", mse, "\nR² Score:", r2, "\n")
MSE: 2656.233
R² Score: 0.9984722
> cat("Final Accuracy (R² Score):", r2 * 100, "%\n")
Final Accuracy (R² Score): 99.84722 %
> *
```

Python

Code :

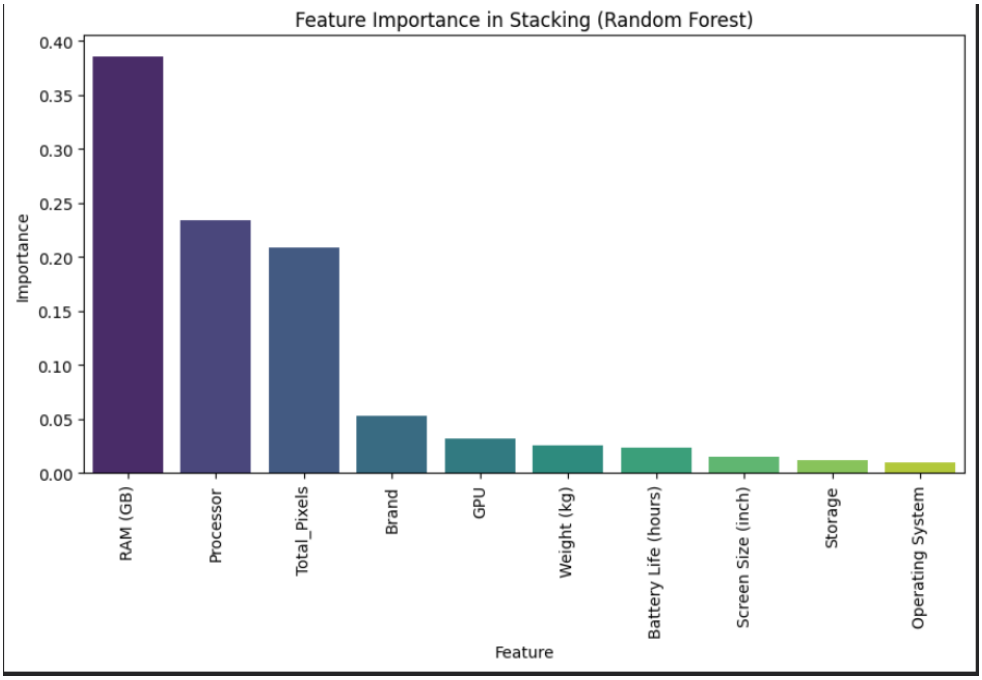
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import StackingRegressor, RandomForestRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset
file_path = r"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop_prices.csv"
df = pd.read_csv(file_path)
# Convert Storage to numerical format
df['Storage'] = df['Storage'].str.extract(r'(\d+)').dropna().astype(float)
# Convert Resolution to total pixel count
df[['Width', 'Height']] = df['Resolution'].str.extract(r'(\d+)x(\d+)').astype(float)
df['Total_Pixels'] = df['Width'] * df['Height']
df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)  # Drop old Resolution column
# Encode categorical variables
```

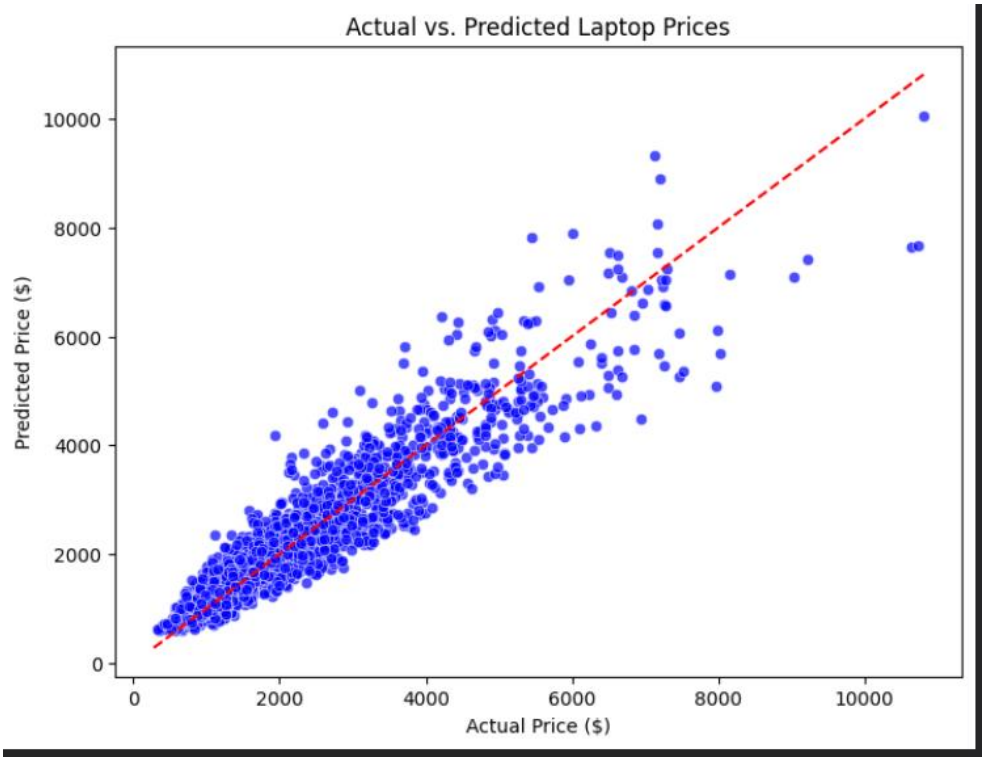
```
for col in ['Brand', 'Processor', 'GPU', 'Operating System']:
    df[col] = LabelEncoder().fit_transform(df[col])
# Define features and target
X = df.drop(columns=['Price ($)'])
y = df['Price ($)']
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Define base models
base_models = [
    ('decision_tree', DecisionTreeRegressor(max_depth=4)),
    ('random_forest', RandomForestRegressor(n_estimators=10, random_state=42)),
        ('adaboost', AdaBoostRegressor(DecisionTreeRegressor(max_depth=4), n_estimators=10, random_state=42))
]
# Meta-model
meta_model = Ridge(alpha=1.0)
# Stacking Regressor
stacking_model = StackingRegressor(estimators=base_models, final_estimator=meta_model)
# Train model
stacking_model.fit(X_train, y_train)
# Predictions
y_pred = stacking_model.predict(X_test)
# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"MSE: {mse:.2f}, R² Score: {r2:.4f}")
print(f"Final Accuracy (R² Score): {r2 * 100:.2f}%")
#  1. Feature Importance (from Random Forest)
feature_importances = stacking_model.named_estimators_['random_forest'].feature_importances_
sorted_indices = np.argsort(feature_importances)[::-1]
plt.figure(figsize=(10, 5))
```

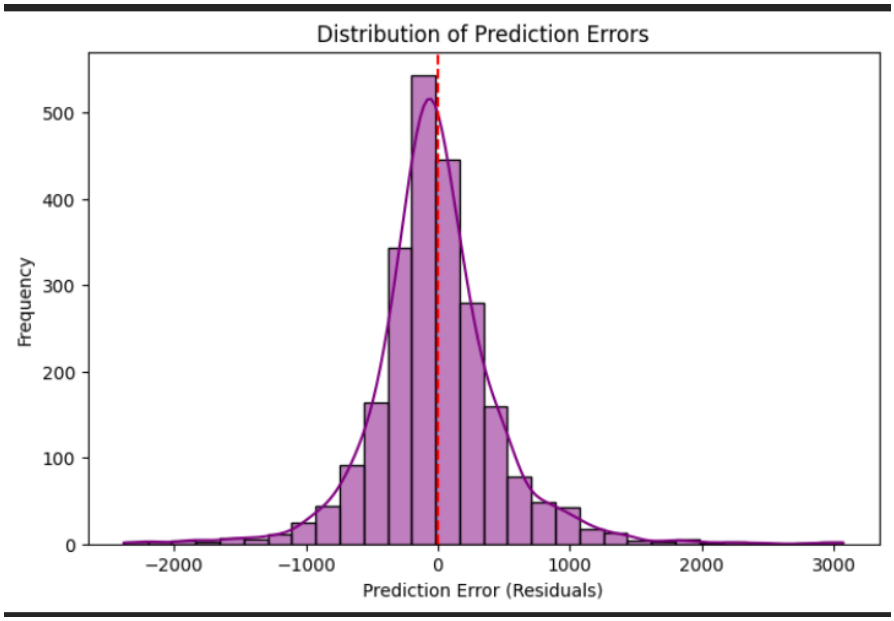
```
sns.barplot(x=X.columns[sorted_indices], y=feature_importances[sorted_indices], palette="viridis")
plt.xticks(rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature Importance in Stacking (Random Forest)")
plt.show()
# 2. Predicted vs. Actual Prices
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.7, color="blue")
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')  # Ideal prediction line
plt.xlabel("Actual Price ($)")
plt.ylabel("Predicted Price ($)")
plt.title("Actual vs. Predicted Laptop Prices")
plt.show()
# 3. Residual Errors Histogram
residuals = y_test - y_pred
plt.figure(figsize=(8, 5))
sns.histplot(residuals, bins=30, kde=True, color="purple")
plt.axvline(0, color='red', linestyle='dashed')  # Mean error line
plt.xlabel("Prediction Error (Residuals)")
plt.ylabel("Frequency")
plt.title("Distribution of Prediction Errors")
plt.show()
```

Output:

```
MSE: 231109.73, R² Score: 0.8696
Final Accuracy (R² Score): 86.96%
```



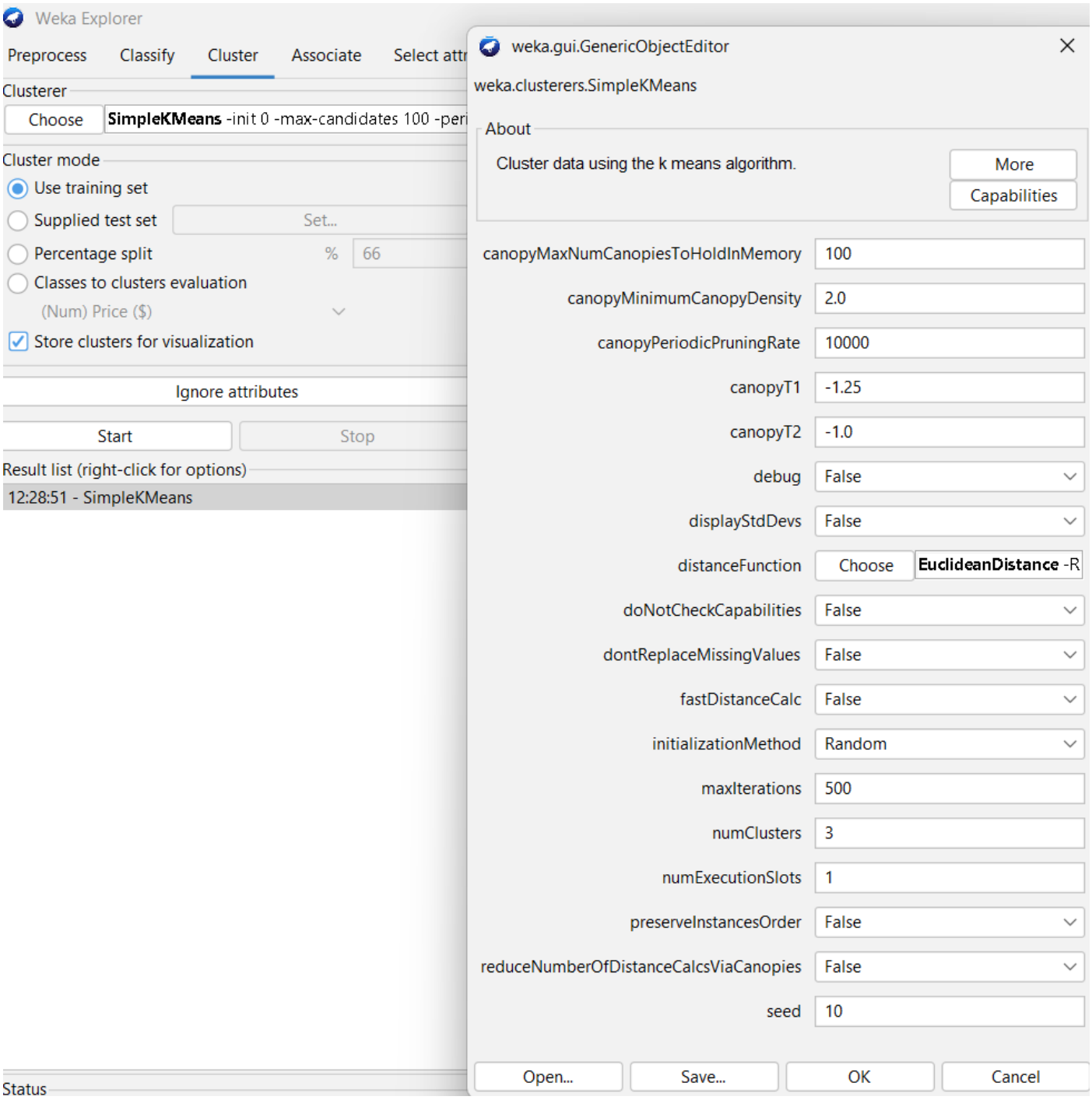

Actual vs. Predicted Laptop Prices
Predicted Price ($)
Actual Price ($)
0
2000
4000
6000
8000
10000


Distribution of Prediction Errors
Frequency
Prediction Error (Residuals)
0
100
200
300
400
500
−2000
−1000
0
1000
2000
3000

Weka :

K MEANS:

OUTPUT:



Clusterer output

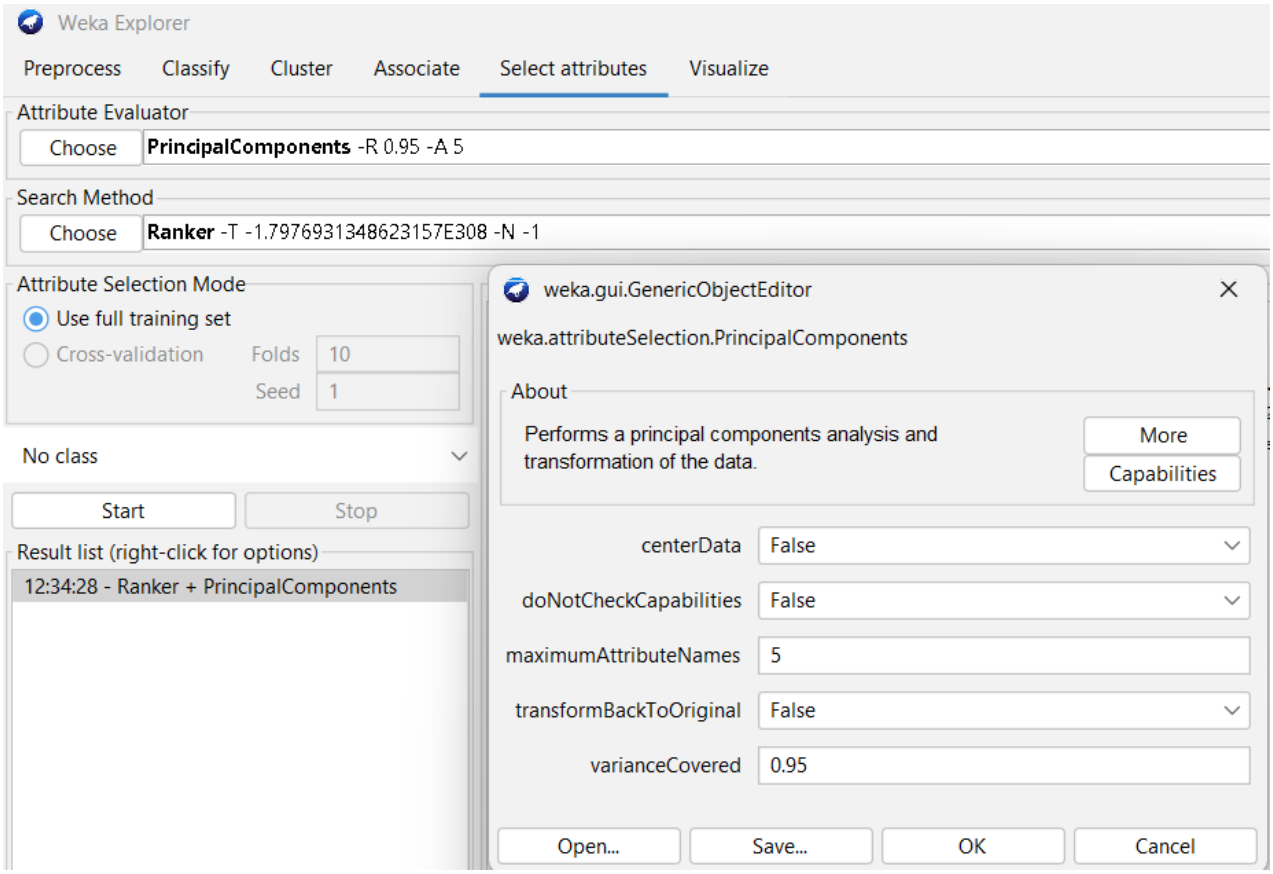
```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.2
Relation:     laptop_prices_sampled_normalized-weka.filters.unsupervised.attribute.Standardize
Instances:    500
Attributes:   11
              Brand
              Processor
              RAM (GB)
              Storage
              GPU
              Screen Size (inch)
              Resolution
              Battery Life (hours)
              Weight (kg)
              Operating System
              Price ($)
Test mode:    evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 10
Within cluster sum of squared errors: 2248.2942315687897

Initial starting points (random):

Cluster 0: HP,AMD_Ryzen_5,1.681194,2TB_SSD,Nvidia_RTX_2060,0.551068,3840x2160,0.264058,-0.836737,Linux,0.377607
Cluster 1: Dell,AMD_Ryzen_7,-1.017063,512GB_SSD,Nvidia_RTX_2060,1.488505,1920x1080,-1.786359,0.127732,Windows,0.094774
Cluster 2: Dell,AMD_Ryzen_3,-0.837179,512GB_SSD,Nvidia_RTX_3080,0.262626,3840x2160,-0.761151,1.046991,FreeDOS,0.128661
```

Clusterer output

```
kMeans
======

Number of iterations: 10
Within cluster sum of squared errors: 2248.2942315687897

Initial starting points (random):

Cluster 0: HP,AMD_Ryzen_5,1.681194,2TB_SSD,Nvidia_RTX_2060,0.551068,3840x2160,0.264058,-0.836737,Linux,0.377607
Cluster 1: Dell,AMD_Ryzen_7,-1.017063,512GB_SSD,Nvidia_RTX_2060,1.488505,1920x1080,-1.786359,0.127732,Windows,0.094774
Cluster 2: Dell,AMD_Ryzen_3,-0.837179,512GB_SSD,Nvidia_RTX_3080,0.262626,3840x2160,-0.761151,1.046991,FreeDOS,0.128661

Missing values globally replaced with mean/mode

Final cluster centroids:
                                        Cluster#
Attribute                  Full Data           0                1                2
                             (500.0)     (164.0)          (167.0)          (169.0)
=============================================================================================
Brand                         Apple          HP             Asus             Dell
Processor                 AMD_Ryzen_9 AMD_Ryzen_5    AMD_Ryzen_7    AMD_Ryzen_3
RAM (GB)                         -0      0.2871          -0.1144          -0.1655
Storage                      2TB_SSD     2TB_SSD         512GB_SSD        512GB_SSD
GPU                       Integrated Nvidia_RTX_2060 AMD_Radeon_RX_6600 Nvidia_RTX_3080
Screen Size (inch)               -0     -0.1305           0.2864          -0.1564
Resolution                 1366x768    3840x2160         1920x1080        1366x768
Battery Life (hours)             -0      0.1246          -0.0674          -0.0543
Weight (kg)                      -0     -0.0359          -0.0699           0.1039
Operating System            Windows       Linux          Windows          FreeDOS
Price ($)                      0.19      0.2382           0.1845           0.1486



Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===
```

PCA

OUTPUT:


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Attribute Evaluator
Choose
PrincipalComponents -R 0.95 -A 5
Search Method
Choose
Ranker -T -1.7976931348623157E308 -N -1
Attribute Selection Mode
Use full training set
Cross-validation
Folds
10
Seed
1
No class
Start
Stop
Result list (right-click for options)
12:34:28 - Ranker + PrincipalComponents
weka.gui.GenericObjectEditor
weka.attributeSelection.PrincipalComponents
About
Performs a principal components analysis and transformation of the data.
More
Capabilities
centerData
False
doNotCheckCapabilities
False
maximumAttributeNames
5
transformBackToOriginal
False
varianceCovered
0.95
Open...
Save...
OK
Cancel

Attribute selection output

```
=== Run information ===

Evaluator:     weka.attributeSelection.PrincipalComponents -R 0.95 -A 5
Search:        weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:      laptop_prices_sampled_normalized-weka.filters.unsupervised.attribute.Standardize
Instances:     500
Attributes:    11
               Brand
               Processor
               RAM (GB)
               Storage
               GPU
               Screen Size (inch)
               Resolution
               Battery Life (hours)
               Weight (kg)
               Operating System
               Price ($)
Evaluation mode:    evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (unsupervised):
        Principal Components Attribute Transformer
```

Attribute selection output

Correlation matrix

1	-0.09	-0.12	-0.11	-0.11	-0.11	-0.11	-0.12	-0.11	-0.11	0.01	-0.07	-0.02	-0.04	0.03	0.11	0	-0.02	0.03	-0.01	-0.03	0.03	-0.03	0.04	0
-0.09	1	-0.1	-0.1	-0.1	-0.09	-0.1	-0.1	-0.09	-0.09	-0.02	0.06	-0.02	0.01	-0.01	0.03	0.01	-0.05	0.03	0.01	-0.05	0.01	0.03	-0.01	-0
-0.12	-0.1	1	-0.12	-0.12	-0.11	-0.12	-0.13	-0.12	-0.12	-0.04	-0.03	0.12	-0.03	0.01	-0.04	0.01	0	0.01	0.03	0.01	-0.03	0.01	-0.03	-0
-0.11	-0.1	-0.12	1	-0.12	-0.11	-0.11	-0.12	-0.11	-0.11	0.02	-0	-0.07	-0	0	-0.01	0.08	-0.03	-0.01	-0.04	0.02	-0.03	0.15	-0.1	-0
-0.11	-0.1	-0.12	-0.12	1	-0.11	-0.12	-0.12	-0.11	-0.11	-0.02	0.07	-0.01	0.01	0.03	-0.03	0.01	-0.07	-0.04	0.02	0.08	-0.02	-0.07	-0.01	0
-0.11	-0.09	-0.11	-0.11	-0.11	1	-0.11	-0.12	-0.11	-0.11	-0.04	0.03	-0.01	0.12	-0.02	-0.02	-0.03	-0.01	0.04	-0.02	0.02	0.03	-0.04	-0	0
-0.11	-0.1	-0.12	-0.11	-0.12	-0.11	1	-0.12	-0.11	-0.11	-0.01	0	0.05	-0.02	0.07	0.02	-0.06	-0.04	-0.04	-0.05	0.04	-0.08	0.06	0.02	-0
-0.12	-0.1	-0.13	-0.12	-0.12	-0.12	-0.12	1	-0.12	-0.12	0.03	-0.05	-0.01	-0.01	0.04	-0.06	-0.03	0.09	0.08	-0.07	0.02	0.03	0	0.01	0
-0.11	-0.09	-0.12	-0.11	-0.11	-0.11	-0.11	-0.12	1	-0.11	0.05	-0.01	-0.04	0.01	-0.06	-0	0.03	0.04	-0.04	0.04	-0.02	-0.04	-0.03	0.04	-0
-0.11	-0.09	-0.12	-0.11	-0.11	-0.11	-0.11	-0.12	-0.11	1	0.03	0.02	0	-0.06	-0.1	0.02	-0.02	0.09	-0.06	0.07	-0.11	0.1	-0.08	0.03	0
0.01	-0.02	-0.04	0.02	-0.02	-0.04	-0.01	0.03	0.05	0.03	1	-0.16	-0.13	-0.14	-0.16	-0.13	-0.14	-0.16	0.02	-0.05	0.05	0	0.01	-0.01	-0
-0.07	0.06	-0.03	-0	0.07	0.03	0	-0.05	-0.01	0.02	-0.16	1	-0.13	-0.14	-0.17	-0.14	-0.15	-0.16	0.01	0.02	-0.01	0.06	-0.09	0.02	0
-0.02	-0.02	0.12	-0.07	-0.01	-0.01	0.05	-0.01	-0.04	0	-0.13	-0.13	1	-0.12	-0.14	-0.11	-0.12	-0.14	0	0.02	0.06	0.03	-0.06	-0.06	0
-0.04	0.01	-0.03	-0	0.01	0.12	-0.02	-0.01	0.01	-0.06	-0.14	-0.14	-0.12	1	-0.15	-0.12	-0.13	-0.15	-0.01	-0.07	-0.03	0.03	0.1	-0.02	-0
0.03	-0.01	0.01	0	0.03	-0.02	0.07	0.04	-0.06	-0.1	-0.16	-0.17	-0.14	-0.15	1	-0.14	-0.15	-0.17	-0.08	0.01	-0.01	-0.05	0.06	-0.01	-0
0.11	0.03	-0.04	-0.01	-0.03	-0.02	0.02	-0.06	-0	0.02	-0.13	-0.14	-0.11	-0.12	-0.14	1	-0.12	-0.14	0.01	0.01	-0.03	-0.05	-0.07	0.13	-0
0	0.01	0.01	0.08	0.01	-0.03	-0.06	-0.03	0.03	-0.02	-0.14	-0.15	-0.12	-0.13	-0.15	-0.12	1	-0.15	0.04	0.06	-0.02	0.01	0.03	-0.09	-0
-0.02	-0.05	0	-0.03	-0.07	-0.01	-0.04	0.09	0.04	0.09	-0.16	-0.16	-0.14	-0.15	-0.17	-0.14	-0.15	1	0.02	-0.01	-0.02	-0.03	0.02	0.03	0
0.03	0.03	0.01	-0.01	-0.04	0.04	-0.04	0.08	-0.04	-0.06	0.02	0.01	0	-0.01	-0.08	0.01	0.04	0.02	1	0.03	0.07	-0.02	-0.04	-0.06	0
-0.01	0.01	0.03	-0.04	0.02	-0.02	-0.05	-0.07	0.04	0.07	-0.05	0.02	0.02	-0.07	0.01	0.01	0.06	-0.01	0.03	1	-0.28	-0.26	-0.24	-0.26	-0
-0.03	-0.05	0.01	0.02	0.08	0.02	0.04	0.02	-0.02	-0.11	0.05	-0.01	0.06	-0.03	-0.01	-0.03	-0.02	-0.02	0.07	-0.28	1	-0.26	-0.25	-0.26	0
0.03	0.01	-0.03	-0.03	-0.02	0.03	-0.08	0.03	-0.04	0.1	0	0.06	0.03	0.03	-0.05	-0.05	0.01	-0.03	-0.02	-0.26	-0.26	1	-0.22	-0.24	-0
-0.03	0.03	0.01	0.15	-0.07	-0.04	0.06	0	-0.03	-0.08	0.01	-0.09	-0.06	0.1	0.06	-0.07	0.03	0.02	-0.04	-0.24	-0.25	-0.22	1	-0.23	-0
0.04	-0.01	-0.03	-0.1	-0.01	-0	0.02	0.01	0.04	0.03	-0.01	0.02	-0.06	-0.02	-0.01	0.13	-0.09	0.03	-0.06	-0.26	-0.26	-0.24	-0.23	1	-0
0.03	-0.06	-0.01	-0	0.01	0.03	-0.01	0.02	-0.05	0.04	-0	0.07	0.03	-0.04	-0.04	-0.06	-0	0.04	0.03	-0.01	0.06	-0.01	-0.02	-0.02	1
0.03	0.01	0.02	0.06	-0.05	-0.02	0.05	-0.01	-0.06	-0.02	0.04	-0.07	-0.02	-0.01	-0	0.01	0.06	0.01	-0.01	0	0.02	-0	-0.06	0.04	-0
-0.06	0.06	-0.03	0.05	0.02	-0.01	-0.08	-0	0.06	0	0.04	0.02	-0.02	-0.04	0.02	0.01	-0.01	-0.04	0.02	-0.02	-0.01	0.09	-0.01	-0.05	-0
0.06	-0.09	0.02	-0.01	0.03	0.09	-0.06	0.04	-0.08	-0	-0.08	-0.01	0.08	-0	0.01	-0.03	-0.01	0.04	-0.11	-0.05	0	0.03	0.02	-0	-0
-0.02	0.06	0.01	-0.06	-0.01	-0.03	0.02	0.04	0	0	0.05	-0.08	-0.02	0.03	0.01	0.03	-0.02	-0	0.04	-0.03	-0.01	-0.04	0.04	0.04	-0
0.01	-0	-0.01	-0.04	-0.01	-0.05	0	0	0.09	0.01	-0.07	0.06	-0.03	0	0.01	0.04	0.04	-0.05	0.08	-0.02	0.01	-0	0.02	-0	-0
-0.05	0.02	0.01	-0	0.01	-0	0.08	-0.08	0.05	-0.03	0.01	0.01	-0.01	0.05	-0.01	-0	-0.05	-0	-0.06	0.13	-0.07	-0.07	0.01	0	-0
-0.01	-0.01	-0.03	0.07	0.06	0.02	0.01	-0.03	-0.08	0	-0.05	-0.03	-0.01	0.01	-0.01	0.07	0.05	-0.02	-0.04	-0.06	-0.05	0.07	0.07	-0.02	0

Attribute selection output

eigenvalue	proportion	cumulative	
2.20644	0.05131	0.05131	-0.608Price ($)-0.389Resolution=3840x2160-0.379RAM (GB)-0.186Processor=AMD_Ryzen_9+0.18 Resolution=1366x768...
1.77141	0.0412	0.09251	-0.398Storage=1TB_HDD+0.361Operating System=macOS+0.292Storage=256GB_SSD-0.287Operating System=FreeDOS-0.264Brand=MSI...
1.73897	0.04044	0.13295	0.468Operating System=Windows-0.359Resolution=1920x1080+0.322Resolution=2560x1440-0.243Brand=Razer-0.242Operating System=Fr
1.70765	0.03971	0.17266	-0.498Operating System=Linux+0.412Resolution=1366x768+0.29 Operating System=macOS-0.248Resolution=1920x1080+0.234Processor=
1.64505	0.03826	0.21092	-0.286Resolution=2560x1440-0.266Processor=AMD_Ryzen_5+0.259Resolution=1366x768+0.245GPU=Nvidia_GTX_1650-0.242Operating Syst
1.59035	0.03698	0.2479	-0.39GPU=AMD_Radeon_RX_6600+0.319Storage=1TB_SSD+0.306Resolution=2560x1440-0.302Resolution=1920x1080+0.279GPU=Nvidia_GTX_16
1.53312	0.03565	0.28356	-0.318Brand=Asus-0.315Storage=1TB_SSD-0.268Processor=Intel_i9-0.266Resolution=1366x768+0.258Storage=1TB_HDD...
1.46109	0.03398	0.31754	0.344Storage=2TB_SSD-0.295Storage=256GB_SSD-0.287GPU=AMD_Radeon_RX_6800+0.278Brand=Dell+0.27 GPU=Integrated...
1.42815	0.03321	0.35075	0.422Brand=Samsung+0.289Storage=512GB_SSD-0.285Brand=Microsoft+0.26 Operating System=FreeDOS+0.227GPU=Nvidia_GTX_1650...
1.42179	0.03306	0.38381	0.424Processor=Intel_i3-0.315GPU=Nvidia_RTX_3080-0.313Processor=AMD_Ryzen_9+0.277Storage=2TB_SSD-0.266Storage=512GB_SSD...
1.36089	0.03165	0.41546	0.397GPU=Nvidia_RTX_2060-0.308Processor=Intel_i5+0.272Processor=AMD_Ryzen_3+0.245Brand=MSI+0.241Storage=1TB_SSD...
1.30891	0.03044	0.4459	0.412GPU=Nvidia_RTX_3060-0.333Storage=512GB_SSD-0.293GPU=AMD_Radeon_RX_6800+0.244Processor=Intel_i5-0.239GPU=Integrated...
1.28654	0.02992	0.47582	0.355GPU=Nvidia_RTX_3060-0.333Processor=Intel_i3+0.328GPU=Integrated+0.306Processor=AMD_Ryzen_7+0.251Screen Size (inch)...
1.24268	0.0289	0.50472	0.363Processor=AMD_Ryzen_7+0.305Brand=Acer-0.301Storage=1TB_HDD+0.262Processor=AMD_Ryzen_5-0.241Brand=Apple...
1.23856	0.0288	0.53353	0.366Processor=Intel_i5-0.311Storage=256GB_SSD+0.292RAM (GB)+0.261Brand=Apple-0.223Resolution=3840x2160...
1.19919	0.02789	0.56141	-0.378GPU=Nvidia_RTX_3060-0.34Processor=AMD_Ryzen_9+0.321Brand=Microsoft-0.277Operating System=Windows+0.273Processor=Intel
1.18125	0.02747	0.58888	-0.403Processor=AMD_Ryzen_3+0.325GPU=Nvidia_GTX_1650-0.289Brand=HP+0.267Resolution=1920x1080+0.232Processor=Intel_i3...
1.14295	0.02658	0.61546	0.385Brand=Lenovo-0.328Screen Size (inch)-0.32Brand=Dell+0.29 GPU=Integrated+0.255Brand=Microsoft...
1.14248	0.02657	0.64203	-0.359Brand=Dell+0.312Processor=Intel_i9-0.297RAM (GB)+0.284Brand=Razer+0.263Storage=1TB_SSD...
1.11282	0.02588	0.66791	0.387Brand=HP-0.304Brand=Asus-0.293GPU=AMD_Radeon_RX_6800+0.274Storage=256GB_SSD-0.25Brand=Lenovo...
1.08511	0.02524	0.69315	0.426Weight (kg)-0.364Brand=HP+0.309Brand=Acer-0.245RAM (GB)-0.218Brand=MSI...
1.05776	0.0246	0.71775	0.491Brand=Acer-0.342Brand=Microsoft+0.314Processor=Intel_i9-0.27Processor=AMD_Ryzen_9-0.242GPU=Nvidia_GTX_1650...
1.0442	0.02428	0.74203	-0.333Processor=AMD_Ryzen_5-0.313Processor=AMD_Ryzen_3-0.288Brand=Samsung+0.266Processor=Intel_i3+0.261Storage=512GB_SSD...
1.01602	0.02363	0.76566	0.435Brand=Razer-0.384Brand=HP-0.259Processor=AMD_Ryzen_9+0.251Brand=Apple+0.204GPU=Nvidia_RTX_3080...
0.99325	0.0231	0.78876	0.347Battery Life (hours)-0.314Processor=Intel_i5-0.291Processor=AMD_Ryzen_5+0.259GPU=AMD_Radeon_RX_6600-0.251GPU=Integrate
0.9801	0.02279	0.81155	0.478Processor=Intel_i7-0.355Storage=256GB_SSD-0.325Brand=Apple-0.283Processor=AMD_Ryzen_3+0.265Brand=Lenovo...
0.94899	0.02207	0.83362	-0.403GPU=AMD_Radeon_RX_6600+0.327GPU=AMD_Radeon_RX_6800-0.289GPU=Integrated+0.251Brand=Microsoft+0.239Weight (kg)...
0.90597	0.02107	0.85469	0.382Brand=MSI+0.358GPU=AMD_Radeon_RX_6800-0.334Brand=Dell-0.288Resolution=3840x2160-0.263Processor=Intel_i3...
0.87716	0.0204	0.87509	0.388Processor=Intel_i5-0.348Screen Size (inch)-0.336Brand=Microsoft+0.261Resolution=3840x2160+0.23 Operating System=macOS.
0.8535	0.01985	0.89494	0.363Brand=Razer-0.334Brand=Acer+0.275Weight (kg)-0.267Screen Size (inch)+0.253Resolution=2560x1440...
0.80778	0.01879	0.91372	0.314GPU=Nvidia_GTX_1650-0.302Brand=MSI-0.299Processor=AMD_Ryzen_7+0.278Brand=Asus+0.257Screen Size (inch)...

Attribute selection output

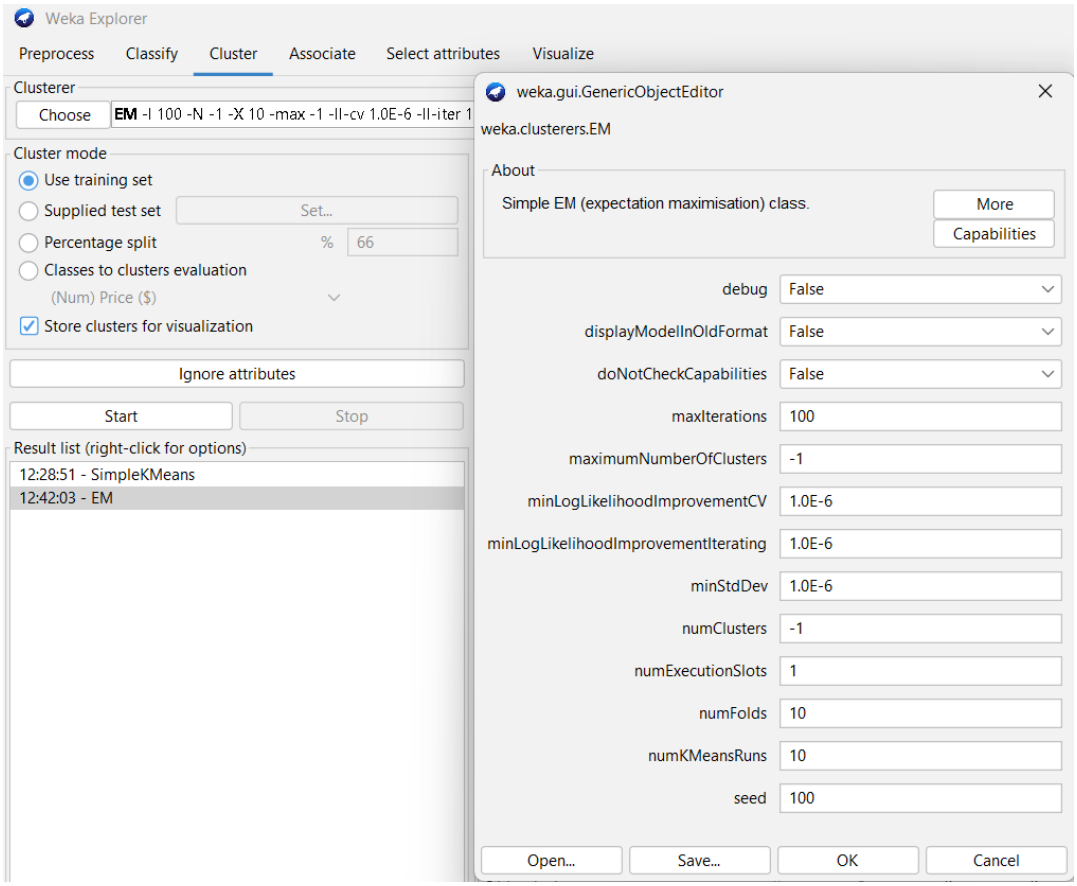
```
Eigenvectors
 V1      V2      V3      V4      V5      V6      V7      V8      V9      V10     V11     V12     V13     V14     V15     V16     V17     V18     V19     V20     V21
 0.092   0.1968  0.0749  0.0651 -0.1842  0.1575  0.2358 -0.0402 -0.0714 -0.0034  0.1459  0.1103 -0.0031  0.0478  0.0092 -0.1202 -0.0636  0.3846 -0.1169 -0.2503 -0.0745
-0.0709 -0.0579  0.0359 -0.0849  0.2409 -0.007  -0.0022 -0.0849  0.0326 -0.0235  0.0679 -0.0519  0.0698  0.3045 -0.0142 -0.194  -0.1318  0.1262 -0.026   0.2256  0.3094
 0.071  -0.0786 -0.1287 -0.0391 -0.1505 -0.1049  0.0079 -0.0255  0.1821 -0.1445  0.0579 -0.1822 -0.1092  0.0982  0.0619 -0.0721 -0.2888 -0.106   0.121   0.3868 -0.3635
-0.0972 -0.2635  0.1474 -0.0098  0.007  -0.0459  0.0952  0.1788 -0.0733 -0.0015  0.2455  0.1775  0.1407 -0.1288 -0.1106  0.2044  0.2302  0.0024 -0.0289  0.083  -0.2182
 0.0764  0.0866 -0.1088 -0.1387  0.0687  0.0591  0.0188  0.2775  0.028  -0.1046 -0.2081 -0.0155 -0.0229 -0.1548 -0.1201 -0.0578  0.1387 -0.3197 -0.3585 -0.1152  0.1815
 0.0074 -0.0315 -0.0028 -0.0806  0.0094  0.0431 -0.1387  0.0419 -0.2852 -0.073  -0.2358  0.1111 -0.101   0.2161  0.0325  0.3211 -0.2234  0.2554 -0.064  -0.0569  0.0872
-0.0774  0.0234 -0.2427  0.1976  0.0924 -0.2054  0.1254  0.1245 -0.1618  0.0132  0.0301 -0.0296  0.0588  0.0177  0.0064 -0.1433  0.0403 -0.0294  0.2838 -0.1613 -0.0607
-0.1576 -0.0197 -0.0355  0.1094 -0.1815  0.2588 -0.087  -0.1279 -0.0662  0.1241 -0.0377 -0.1619 -0.1156 -0.2408  0.2609 -0.0625  0.1399 -0.0986 -0.0478  0.199   0.1675
 0.0047  0.07    0.14    0.0151  0.0652 -0.0927  0.0616 -0.1675  0.4215  0.0918 -0.2329  0.2367 -0.0201 -0.0608  0.0066  0.1505  0.1127 -0.0515  0.1847 -0.0126  0.1226
 0.1511  0.0761  0.1364 -0.0458  0.0751 -0.0744 -0.3185 -0.1931 -0.0108  0.121   0.1722 -0.1811  0.1186 -0.0405 -0.1466 -0.0347  0.0204 -0.1184  0.0594 -0.3043 -0.1102
 0.0924 -0.0377  0.0317  0.0462  0.1346  0.0154  0.0582  0.0767  0.1689  0.4243  0.0512  0.0365 -0.3328 -0.0543  0.0579 -0.0645  0.2323  0.227   0.0949 -0.0808 -0.0863
-0.1044  0.2154 -0.0924 -0.1405  0.1665  0.08   -0.2682  0.1744 -0.0301 -0.1254 -0.1306  0.0701  0.2455 -0.1398 -0.0761  0.0869  0.0109  0.0483  0.3119  0.133  -0.0582
 0.1024 -0.0065 -0.1673 -0.0919 -0.2662 -0.1175 -0.1222  0.0401  0.0599 -0.1198  0.0539 -0.0928 -0.1742  0.2622  0.0291 -0.1552  0.1005 -0.0141  0.1776  0.0102 -0.1499
 0.0641 -0.1515  0.0474 -0.0254  0.0876 -0.0203  0.0742 -0.0748 -0.1977 -0.0712 -0.3081  0.2443 -0.0595  0.2162  0.3658  0.1168 -0.0747 -0.1223 -0.0368  0.0189  0.1022
-0.1865 -0.0694 -0.0331  0.1862  0.0105  0.0877  0.1684  0.0709 -0.081  -0.3127 -0.0288 -0.1944 -0.1385 -0.177  -0.2125 -0.3402 -0.0079  0.0343  0.0074 -0.0775  0.1726
-0.0035  0.197  -0.0417 -0.0354  0.0057  0.0497  0.246  -0.1841 -0.0255  0.059   0.0926  0.0046  0.3063  0.3628 -0.2138  0.0471  0.1526  0.0514 -0.2011 -0.015  -0.1638
 0.0385 -0.0901  0.1379 -0.2155  0.0115 -0.0261  0.1021  0.1007  0.1442 -0.0026  0.2719  0.1456  0.0609 -0.1808  0.0417  0.0371 -0.4033 -0.1557 -0.219  -0.0948  0.0321
 0.0256 -0.0495  0.1018  0.234  -0.174  -0.0838 -0.2358 -0.2154 -0.0317  0.1496  0.0132 -0.1884  0.1001 -0.1987  0.0243  0.2727 -0.0057 -0.0794 -0.1463  0.1005  0.1114
-0.3786  0.1135 -0.0357 -0.0353 -0.0876  0.0514 -0.183   0.0019  0.0044  0.1369  0.0959  0.0625  0.0272  0.0726  0.2921 -0.0123 -0.0618  0.1673 -0.2967  0.0704 -0.245
-0.05    0.2257  0.1662 -0.116  -0.0277 -0.2627  0.0028  0.0879  0.2888 -0.2658  0.1161 -0.3331  0.0025  0.0012  0.1308  0.1485 -0.0539  0.1266 -0.1396 -0.0876  0.1603
-0.1045 -0.1364 -0.2121  0.0147 -0.2419  0.0105 -0.0485  0.3437  0.1367  0.2766 -0.2274  0.1671 -0.1097  0.1406 -0.2212 -0.0415 -0.0287 -0.0571 -0.1753 -0.0428 -0.0765
 0.0912 -0.0014  0.0736 -0.0919  0.0801  0.3188 -0.3154 -0.0323 -0.1298 -0.0712  0.2409  0.1111 -0.0942  0.1721  0.1992 -0.1054  0.0823 -0.1849  0.2628 -0.2113  0.0612
 0.0122 -0.3976  0.028   0.0146  0.1126 -0.0756  0.258  -0.1317 -0.1251 -0.102   0.0337  0.0552  0.1272 -0.3011  0.2189 -0.0413  0.0912  0.1957  0.0505  0.0759 -0.1438
 0.0597  0.2915 -0.0494  0.1828  0.0962  0.0172  0.115  -0.2954 -0.1955  0.1537 -0.1534  0.0069  0.085  -0.0322 -0.3113  0.0337 -0.083  -0.0749  0.0199  0.274  -0.0093
 0.1257 -0.043   0.0358 -0.0735 -0.16    0.0508 -0.1197  0.27   -0.0567  0.2542 -0.1589 -0.2391  0.3276 -0.1243  0.0215  0.0187 -0.1635  0.29    0.1847 -0.0298  0.0226
-0.0464  0.0846  0.0371  0.0337 -0.0908 -0.0629  0.1693  0.0893 -0.1598  0.1619  0.3968  0.147  -0.2414  0.0407 -0.115   0.1242 -0.2271 -0.2262  0.1605  0.232   0.2109
-0.0242 -0.0872  0.0357 -0.0403  0.2451  0.2791 -0.1293  0.0548  0.2271 -0.0558  0.1502  0.0112 -0.0766  0.1091 -0.135   0.0572  0.3252  0.1222 -0.1374  0.2452  0.0666
 0.117  -0.0602 -0.0567 -0.0462 -0.233   0.125  -0.065  -0.1492 -0.1843 -0.3146 -0.0838  0.0447 -0.2244 -0.1613 -0.2187  0.1667  0.0089  0.0469 -0.079  -0.1453 -0.1682
-0.0922 -0.0585 -0.1386 -0.0662  0.0979  0.0127  0.1814 -0.2867  0.0039  0.2259 -0.0945 -0.2934 -0.0944  0.0941  0.1801 -0.0545 -0.0399 -0.136  -0.1029 -0.293  -0.0055
-0.0997  0.0723 -0.035   0.0363 -0.0605  0.0047 -0.0539 -0.1275  0.1965 -0.0945 -0.0634  0.4116  0.3551 -0.0967  0.1088 -0.3778 -0.1237 -0.1027  0.0073 -0.08   -0.0044
 0.0191  0.0868  0.1113  0.151   0.1892 -0.3899  0.015   0.1207 -0.0275 -0.201  -0.1482 -0.0648 -0.0594  0.128   0.1452  0.0626  0.221  -0.0017 -0.0465  0.055  -0.1316
```

Attribute selection output

```
Ranked attributes:
 0.9487    1 -0.608Price ($)-0.389Resolution=3840x2160-0.379RAM (GB)-0.186Processor=AMD_Ryzen_9+0.18 Resolution=1366x768...
 0.9075    2 -0.398Storage=1TB_HDD+0.361Operating System=macOS+0.292Storage=256GB_SSD-0.287Operating System=FreeDOS-0.264Brand=MSI...
 0.8671    3 0.468Operating System=Windows-0.359Resolution=1920x1080+0.322Resolution=2560x1440-0.243Brand=Razer-0.242Operating System=FreeDOS...
 0.8273    4 -0.498Operating System=Linux+0.412Resolution=1366x768+0.29 Operating System=macOS-0.248Resolution=1920x1080+0.234Processor=Intel_i7...
 0.7891    5 -0.286Resolution=2560x1440-0.266Processor=AMD_Ryzen_5+0.259Resolution=1366x768+0.245GPU=Nvidia_GTX_1650-0.242Operating System=FreeDOS...
 0.7521    6 -0.39GPU=AMD_Radeon_RX_6600+0.319Storage=1TB_SSD+0.306Resolution=2560x1440-0.302Resolution=1920x1080+0.279GPU=Nvidia_GTX_1650...
 0.7164    7 -0.318Brand=Asus-0.315Storage=1TB_SSD-0.268Processor=Intel_i9-0.266Resolution=1366x768+0.258Storage=1TB_HDD...
 0.6825    8 0.344Storage=2TB_SSD-0.295Storage=256GB_SSD-0.287GPU=AMD_Radeon_RX_6800+0.278Brand=Dell+0.27 GPU=Integrated...
 0.6493    9 0.422Brand=Samsung+0.289Storage=512GB_SSD-0.285Brand=Microsoft+0.26 Operating System=FreeDOS+0.227GPU=Nvidia_GTX_1650...
 0.6162   10 0.424Processor=Intel_i3-0.315GPU=Nvidia_RTX_3080-0.313Processor=AMD_Ryzen_9+0.277Storage=2TB_SSD-0.266Storage=512GB_SSD...
 0.5845   11 0.397GPU=Nvidia_RTX_2060-0.308Processor=Intel_i5+0.272Processor=AMD_Ryzen_3+0.245Brand=MSI+0.241Storage=1TB_SSD...
 0.5541   12 0.412GPU=Nvidia_RTX_3060-0.333Storage=512GB_SSD-0.293GPU=AMD_Radeon_RX_6800+0.244Processor=Intel_i5-0.239GPU=Integrated...
 0.5242   13 0.355GPU=Nvidia_RTX_3060-0.333Processor=Intel_i3+0.328GPU=Integrated+0.306Processor=AMD_Ryzen_7+0.251Screen Size (inch)...
 0.4953   14 0.363Processor=AMD_Ryzen_7+0.305Brand=Acer-0.301Storage=1TB_HDD+0.262Processor=AMD_Ryzen_5-0.241Brand=Apple...
 0.4665   15 0.366Processor=Intel_i5-0.311Storage=256GB_SSD+0.292RAM (GB)+0.261Brand=Apple-0.223Resolution=3840x2160...
 0.4386   16 -0.378GPU=Nvidia_RTX_3060-0.34Processor=AMD_Ryzen_9+0.321Brand=Microsoft-0.277Operating System=Windows+0.273Processor=Intel_i7...
 0.4111   17 -0.403Processor=AMD_Ryzen_3+0.325GPU=Nvidia_GTX_1650-0.289Brand=HP+0.267Resolution=1920x1080+0.232Processor=Intel_i3...
 0.3845   18 0.385Brand=Lenovo-0.328Screen Size (inch)-0.32Brand=Dell+0.29 GPU=Integrated+0.255Brand=Microsoft...
 0.358    19 -0.359Brand=Dell+0.312Processor=Intel_i9-0.297RAM (GB)+0.284Brand=Razer+0.263Storage=1TB_SSD...
 0.3321   20 0.387Brand=HP-0.304Brand=Asus-0.293GPU=AMD_Radeon_RX_6800+0.274Storage=256GB_SSD-0.25Brand=Lenovo...
 0.3069   21 0.426Weight (kg)-0.364Brand=HP+0.309Brand=Acer-0.245RAM (GB)-0.218Brand=MSI...
 0.2823   22 0.491Brand=Acer-0.342Brand=Microsoft+0.314Processor=Intel_i9-0.27Processor=AMD_Ryzen_9-0.242GPU=Nvidia_GTX_1650...
 0.258    23 -0.333Processor=AMD_Ryzen_5-0.313Processor=AMD_Ryzen_3-0.288Brand=Samsung+0.266Processor=Intel_i3+0.261Storage=512GB_SSD...
 0.2343   24 0.435Brand=Razer-0.384Brand=HP-0.259Processor=AMD_Ryzen_9+0.251Brand=Apple+0.204GPU=Nvidia_RTX_3080...
 0.2112   25 0.347Battery Life (hours)-0.314Processor=Intel_i5-0.291Processor=AMD_Ryzen_5+0.259GPU=AMD_Radeon_RX_6600-0.251GPU=Integrated...
 0.1884   26 0.478Processor=Intel_i7-0.355Storage=256GB_SSD-0.325Brand=Apple-0.283Processor=AMD_Ryzen_3+0.265Brand=Lenovo...
 0.1664   27 -0.403GPU=AMD_Radeon_RX_6600+0.327GPU=AMD_Radeon_RX_6800-0.289GPU=Integrated+0.251Brand=Microsoft+0.239Weight (kg)...
 0.1453   28 0.382Brand=MSI+0.358GPU=AMD_Radeon_RX_6800-0.334Brand=Dell-0.288Resolution=3840x2160-0.263Processor=Intel_i3...
 0.1249   29 0.388Processor=Intel_i5-0.348Screen Size (inch)-0.336Brand=Microsoft+0.261Resolution=3840x2160+0.23 Operating System=macOS...
 0.1051   30 0.363Brand=Razer-0.334Brand=Acer+0.275Weight (kg)-0.267Screen Size (inch)+0.253Resolution=2560x1440...
 0.0863   31 0.314GPU=Nvidia_GTX_1650-0.302Brand=MSI-0.299Processor=AMD_Ryzen_7+0.278Brand=Asus+0.257Screen Size (inch)...
 0.0676   32 -0.362Battery Life (hours)-0.339GPU=Nvidia_RTX_3080+0.288Brand=Asus+0.272GPU=AMD_Radeon_RX_6600+0.27 Storage=2TB_SSD...
 0.0497   33 -0.368Brand=Lenovo+0.341Processor=AMD_Ryzen_7-0.338Processor=AMD_Ryzen_5+0.267GPU=Nvidia_RTX_3080-0.264Operating System=Linux...

Selected attributes: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33 : 33
```

EM:

OUTPUT:


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
EM -I 100 -N -1 -X 10 -max -1 -ll-cv 1.0E-6 -ll-iter 1
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Num) Price ($)
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
12:28:51 - SimpleKMeans
12:42:03 - EM
weka.gui.GenericObjectEditor
weka.clusterers.EM
About
Simple EM (expectation maximisation) class.
More
Capabilities
debug
False
displayModelInOldFormat
False
doNotCheckCapabilities
False
maxIterations
100
maximumNumberOfClusters
-1
minLogLikelihoodImprovementCV
1.0E-6
minLogLikelihoodImprovementIterating
1.0E-6
minStdDev
1.0E-6
numClusters
-1
numExecutionSlots
1
numFolds
10
numKMeansRuns
10
seed
100
Open...
Save...
OK
Cancel

Clusterer output

```
EM
==

Number of clusters selected by cross validation: 6
Number of iterations performed: 9
```

Attribute	Cluster 0 (0.21)	1 (0.12)	2 (0.09)	3 (0.14)	4 (0.31)	5 (0.12)
Brand						
Lenovo	16.8318	3.6641	3.7704	11.2587	15.5937	3.8814
Acer	5.7941	9.041	7.3768	5.4378	8.9917	7.3587
HP	11.0119	3.4587	8.2094	8.294	21.3958	8.6302
MSI	5.0689	13.8023	3.9513	7.1192	13.8751	14.1832
Dell	16.7859	1.168	2.7842	11.5901	23.2891	4.3827
Microsoft	8.7421	6.0637	7.0979	7.7259	18.4116	4.9587
Razer	15.6176	11.5521	1.7328	4.7503	12.2936	10.0535
Apple	8.7889	18.4367	6.5229	11.7814	7.9159	9.5541
Samsung	13.0503	3.6871	9.7024	4.7041	19.2774	4.5786
Asus	11.9146	1.1174	6.0842	8.3171	23.3307	4.2359
[total]	113.6062	71.9912	57.2322	80.9787	164.3747	71.8171
Processor						
Intel_i3	10.7211	1.9044	9.1328	13.2738	25.7555	11.2124
Intel_i9	28.8175	17.2019	5.197	9.6574	6.6751	7.4511
AMD_Ryzen_5	5.1812	1.3403	5.7115	11.4353	24.5961	6.7355
Intel_i5	7.0476	6.599	9.0887	7.6272	27.923	5.7145
AMD_Ryzen_9	22.2918	25.1743	1.3938	4.7384	18.2481	9.1537
AMD_Ryzen_7	22.1659	7.8024	2.8738	8.5346	12.2814	4.3419
AMD_Ryzen_3	2.4833	3.1632	11.3687	12.2844	23.6602	12.0403
Intel_i7	12.8978	6.8057	10.4659	11.4277	23.2353	13.1676
[total]	111.6062	69.9912	55.2322	78.9787	162.3747	69.8171
RAM (GB)						
mean	0.2576	0.6136	0.5854	0.9984	0.0717	0.229
std. dev.	0.1835	0.3449	0.305	0.0292	0.0777	0.186

Clusterer output

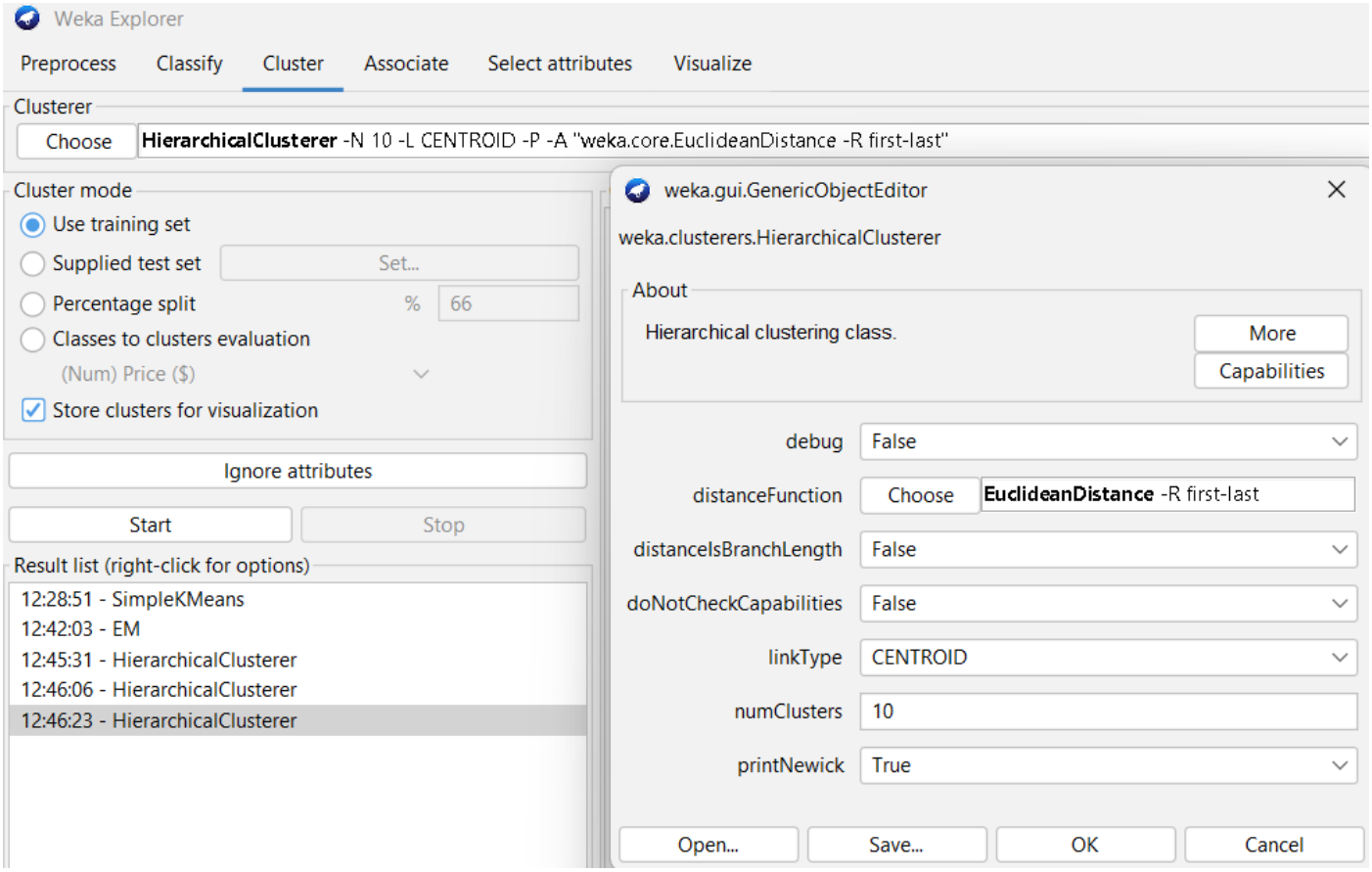
```
=== Run information ===

Scheme:       weka.clusterers.EM -I 100 -N -1 -X 10 -max -1 -ll-cv 1.0E-6 -ll-iter 1.0E-6 -M 1.0E-6 -K 10 -num-slots 1 -S 100
Relation:     laptop_prices_sampled_normalized-weka.filters.unsupervised.attribute.Standardize-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0
Instances:    500
Attributes:   11
              Brand
              Processor
              RAM (GB)
              Storage
              GPU
              Screen Size (inch)
              Resolution
              Battery Life (hours)
              Weight (kg)
              Operating System
              Price ($)
Test mode:    evaluate on training data


=== Clustering model (full training set) ===
```

GPU						
Integrated	16.2496	2.5515	7.5733	16.2042	23.2313	18.19
Nvidia_RTX_2060	12.0439	13.0319	5.6834	10.4557	24.9426	13.8424
Nvidia_GTX_1650	14.3798	8.4995	11.3333	14.7152	19.0542	9.0181
Nvidia_RTX_3080	12.407	7.0622	4.6042	6.564	30.1781	7.1844
AMD_Radeon_RX_6800	16.0549	13.6955	6.7378	10.1955	17.1597	12.1566
Nvidia_RTX_3060	17.663	13.0724	10.4734	13.0748	16.7086	3.0078
AMD_Radeon_RX_6600	21.8078	11.0781	7.8268	6.7693	30.1002	5.4178
[total]	110.6062	68.9912	54.2322	77.9787	161.3747	68.8171
Screen Size (inch)						
mean	0.4628	0.5115	0.4196	0.4787	0.4538	0.6223
std. dev.	0.3563	0.3363	0.3365	0.3355	0.3472	0.318
Resolution						
2560x1440	42.9181	11.8895	7.9448	11.1477	35.5672	25.5327
1920x1080	23.6587	7.0818	11.3048	28.1798	44.1989	9.576
1366x768	20.3287	4.713	17.639	20.6322	60.025	16.6621
3840x2160	20.7007	42.3069	14.3436	15.019	18.5836	14.0462
[total]	107.6062	65.9912	51.2322	74.9787	158.3747	65.8171
Battery Life (hours)						
mean	0.5177	0.5394	0.3416	0.5465	0.5052	0.6953
std. dev.	0.2708	0.2884	0.2273	0.2773	0.2783	0.2252
Weight (kg)						
mean	0.4443	0.5043	0.5182	0.4974	0.4721	0.5256
std. dev.	0.2808	0.299	0.2732	0.2803	0.2998	0.2687
Operating System						
macOS	42.4275	15.0315	5.7693	21.3718	36.2238	6.1761
Linux	28.1692	5.7183	16.3414	16.5979	40.0005	16.1726
FreeDOS	12.9637	23.3952	14.004	19.5029	38.2715	21.8628
Windows	24.0457	21.8462	15.1175	17.5061	43.8789	21.6056
[total]	107.6062	65.9912	51.2322	74.9787	158.3747	65.8171
Price ($)						
mean	0.18	0.4294	0.1725	0.2678	0.0881	0.1449
std. dev.	0.0483	0.1485	0.0591	0.0782	0.0309	0.0566

```
Time taken to build model (full training data) : 1.51 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       102 ( 20%)
1        54 ( 11%)
2        29 (  6%)
3        89 ( 18%)
4       172 ( 34%)
5        54 ( 11%)


Log likelihood: -9.6877
```

HIERACHICAL CUSTERING:

OUTPUT:



Clusterer output

```
=== Run information ===

Scheme:       weka.clusterers.HierarchicalClusterer -N 10 -L CENTROID -P -A "weka.core.EuclideanDistance -R first-last"
Relation:     laptop_prices_sampled_normalized-weka.filters.unsupervised.attribute.Standardize-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0
Instances:    500
Attributes:   11
              Brand
              Processor
              RAM (GB)
              Storage
              GPU
              Screen Size (inch)
              Resolution
              Battery Life (hours)
              Weight (kg)
              Operating System
              Price ($)
Test mode:    evaluate on training data


=== Clustering model (full training set) ===

Cluster 0
((((((((((((((0.091275:1.54638,0.118097:1.54638):-0.21805,0.268264:1.32833):-0.02019,0.391072:1.30815):0.1271,(0.142006:0.98089,0.286849:0.98089):0.45435):0.0

Cluster 1
((((((((((0.113002:1.52722,0.114619:1.52722):0.18962,(0.278461:1.772,0.158463:1.772):-0.05516):0.04412,(((0.180744:1.48771,0.276476:1.48771):-0.03049,0.233886:1

Cluster 2
(0.435868:1.83276,0.19444:1.83276)

Cluster 6
(0.236424:1.74802,0.277164:1.74802)

Cluster 7
(0.269956:1.98178,0.158975:1.98178)
```

```
Time taken to build model (full training data) : 0.17 seconds

=== Model and evaluation on training set ===

Clustered Instances

0        38 (  8%)
1       451 ( 90%)
2         2 (  0%)
3         1 (  0%)
4         1 (  0%)
5         1 (  0%)
6         2 (  0%)
7         2 (  0%)
8         1 (  0%)
9         1 (  0%)
```

LINEAR REGRESSION:

```
Time taken to build model: 0.03 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9321
Mean absolute error                      0.0315
Root mean squared error                  0.0467
Relative absolute error                 33.1371 %
Root relative squared error             36.1366 %
Total Number of Instances              500
```

STACKING:

```
Stacking

Base classifiers

ZeroR predicts class value: 0.1899751920000002



Meta classifier

ZeroR predicts class value: 0.1899751920000002

Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                 -0.1554
Mean absolute error                      0.0951
Root mean squared error                  0.1291
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances              500
```

RANDOM FOREST

```
=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.11 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.02 seconds

=== Summary ===

Correlation coefficient                  0.9939
Mean absolute error                      0.0184
Root mean squared error                  0.0271
Relative absolute error                 19.4052 %
Root relative squared error             21.072  %
Total Number of Instances              500
```

BAGGING:

```
=== Classifier model (full training set) ===

Bagging with 10 iterations and base learner

weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

Time taken to build model: 0.05 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.9281
Mean absolute error                      0.0358
Root mean squared error                  0.0528
Relative absolute error                 37.7681 %
Root relative squared error             40.9949 %
Total Number of Instances              500
```

GAUSSIAN PROCESS:

```
Gaussian Processes

Kernel used:
  Linear Kernel: K(x,y) = <x,y>

All values shown based on: Normalize training data

Average Target Value : 0.17968350120613205
Inverted Covariance Matrix:
    Lowest Value = -0.06485963733730112
    Highest Value = 0.9437279598048892
Inverted Covariance Matrix * Target-value Vector:
    Lowest Value = -0.07602076195594497
    Highest Value = 0.39634589967639744



Time taken to build model: 0.14 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.22 seconds

=== Summary ===

Correlation coefficient                  0.9419
Mean absolute error                      0.0292
Root mean squared error                  0.0433
Relative absolute error                 30.7825 %
Root relative squared error             33.6268 %
Total Number of Instances              500
```

Result :

Support Vector Machine (SVM) and Random Forest achieved the highest accuracy in predicting laptop prices. K-Means effectively grouped laptops into meaningful clusters based on specifications